Labelled Superposition for PLTL

Martin Suda[1,2,3,⋆] and Christoph Weidenbach[1,⋆⋆]

[1] Max-Planck-Institut für Informatik, Saarbrücken, Germany
[2] Saarland University, Saarbrücken, Germany
[3] Charles University, Prague, Czech Republic

Abstract. This paper introduces a new decision procedure for PLTL based on labelled superposition. Its main idea is to treat temporal formulas as infinite sets of purely propositional clauses over an extended signature. These infinite sets are then represented by finite sets of labelled propositional clauses. The new representation enables the replacement of the complex temporal resolution rule, suggested by existing resolution calculi for PLTL, by a fine grained repetition check of finitely saturated labelled clause sets followed by a simple inference. The completeness argument is based on the standard model building idea from superposition. It inherently justifies ordering restrictions, redundancy elimination and effective partial model building. The latter can be directly used to effectively generate counterexamples of non-valid PLTL conjectures out of saturated labelled clause sets in a straightforward way.

1 Introduction

Propositional linear temporal logic [15] is an extension of classical propositional logic for reasoning about time. It introduces temporal operators such as $\Diamond P$ meaning P holds *eventually* in the future, $\Box P$ meaning P holds *always* in the future, and $\bigcirc P$ meaning P holds at the *next* time point. Time is considered to be a linear discrete sequence of time points represented by propositional valuations, called *worlds*. Such a potentially infinite sequence forms a PLTL interpretation. A decision procedure for PLTL takes a PLTL formula P and checks whether it is valid, i.e., that all PLTL interpretations are actually models for P. For example, the PLTL formula $\Box P \rightarrow \bigcirc P$ is valid (a theorem) whereas the PLTL formula $\bigcirc P \rightarrow \Box P$ is not, but is satisfiable, i.e., there is a PLTL model for it.

Attempts to use clausal resolution to attack the decision problem for PLTL appeared first in [3, 21]. The most recent resolution-based approach is the one of [6]. It relies on a satisfiability preserving clausal translation of PLTL formulas, where, in particular, all nestings of temporal operators are reduced to formulas (and, eventually, clauses) of the form P, $\Box(P \rightarrow \bigcirc Q)$, and $\Box(P \rightarrow \Diamond Q)$, where P and Q do not contain temporal operators. Classical propositional resolution is extended to cope with "local" temporal reasoning within neighbouring worlds,

⋆ Supported by Microsoft Research through its PhD Scholarship Programme.
⋆⋆ Supported by the German Transregional Collaborative Research Center SFB/TR 14 AVACS.

while an additional inference rule called *temporal resolution* is introduced to deal with *eventuality* ($\Box\Diamond$) clauses. The temporal resolution rule is quite complex. It requires a search for certain combinations of clauses that together form a *loop*, i.e. imply that certain sets of worlds must be discarded from consideration, because an eventuality clause would be unsatisfied forever within them. This is verified via an additional proof task. Finally, the conclusion of the rule needs to be transformed back into the clause form.

Our labelled superposition calculus builds on a refinement of the above clause normal form [4]. It introduces a notion of labelled clauses in the spirit of [12] and replaces the temporal resolution rule by saturation and a new *Leap* rule. Although in PLTL equality is not present, the principles of superposition are fundamental for our calculus. Our completeness result is based on a model generation approach with an inherent redundancy concept based on a total well-founded ordering on the propositional atoms.

The main contributions of our paper are: 1) we replace the temporal resolution rule by a much more streamlined saturation of certain labelled clauses followed by a simple Leap inference, 2) our inference rules are guided by an ordering restriction that is known to reduce the search space considerably, 3) the completeness proof justifies an abstract redundancy notion that enables strong reductions, 4) if a contradiction cannot be derived, a temporal model can be extracted from a saturated clause set.

The paper is organized as follows. We fix our notation and formalize the problem to be solved in Sect. 2. Then in Sect. 3 we show how to use labelled clauses as a tool to "lift" the standard propositional calculus to reason about PLTL-satisfiability. Our calculus is introduced in Sect. 4 and used as a basis for an effective decision procedure in Sect. 5. We deal with abstract redundancy, its relation to the completeness proof, and model building in Sect. 6. Discussion of previous work and an experimental comparison to existing resolution approaches appear in Sect. 7. Finally, Sect. 8 concludes. Detailed proofs and additional material are available in a technical report [20].

2 Preliminaries

In this section we fix our notation and briefly recall the standard notions we will be using. We assume the reader to be familiar with ordered resolution calculus for propositional logic and its completeness proof [2] including the concepts of redundancy, saturation, and model construction. In the following, the symbol $\mathbb{N}$ stands for the naturals, and $\mathbb{N}^+$ denotes the set $\mathbb{N} \setminus \{0\}$.

The language of propositional formulas and clauses over a given signature $\Sigma = \{p, q, \ldots\}$ of propositional variables is defined in the usual way. We denote propositional clauses by letters C or D, possibly with subscripts, and understand them as multisets of literals. By propositional *valuation*, or simply a *world*, we mean a mapping $V : \Sigma \to \{0, 1\}$. We write $V \models P$ if a propositional formula P is satisfied by V. The semantics of PLTL is based on a discrete linear model of time, where the structure of possible time points is isomorphic to $\mathbb{N}$. A *PLTL-interpretation* is a sequence $(V_i)_{i \in \mathbb{N}}$ of propositional valuations.

In order to be able to talk about several neighbouring worlds at once we introduce copies (i.e. pairwise disjoint, bijectively equivalent sets) of the basic signature Σ. We use priming to denote the shift from one signature to the next (thus Σ' is the set of symbols $\{p', q', \dots\}$), and shorten repeated primes by parenthesised integers (e.g. p''' is the same thing as $p^{(3)}$). This notational convention can be extended from symbols and signatures to formulas, and also to valuations in a natural way. For example, if V is a valuation over $\Sigma^{(i)}$ we write V' for the valuation over $\Sigma^{(i+1)}$ such that $V'(p^{(i+1)}) = V(p^{(i)})$ for every $p \in \Sigma$. We also need to consider formulas over two consecutive joined signatures, e.g. over $\Sigma \cup \Sigma'$. Such formulas can be evaluated over the respective joined valuations. When both V_1 and V_2 are valuations over Σ, we write $[V_1, V_2]$ as a shorthand for the mapping $V_1 \cup (V_2)' : (\Sigma \cup \Sigma') \to \{0, 1\}$.

As usual in refutational theorem proving, our method starts with negating the input formula and translating the result into a clause normal form. Then the task is to show that the translated form is unsatisfiable, which implies validity of the original formula. We build on the *Separated Normal Form* to which any PLTL formula can be translated by a satisfiability preserving transformation with at most linear increase in size [6, 4]. We skip here the details of the transformation due to lack of space and instead directly present its final result, the starting point for our method:

Definition 1. *A* PLTL-specification S *is a quadruple* (Σ, I, T, G) *such that*

- Σ *is a finite propositional signature,*
- I *is a set of* initial *clauses* C_i *(over the signature* Σ*),*
- T *is a set of* step *clauses* $C_t \vee D_t'$ *(over the joint signature* $\Sigma \cup \Sigma'$*),*
- G *is a set of* goal *clauses* C_g *(over the signature* Σ*).*

The initial and step clauses match their counterparts from [6] in the obvious way. Our goal clauses are a generalization of a single unconditional *sometimes* clause that can be obtained using the transformations described in [4]. The whole specification represents the PLTL formula:

$$\left(\bigwedge C_i\right) \wedge \Box \left(\bigwedge (C_t \vee \bigcirc D_t)\right) \wedge \Box\Diamond \left(\bigwedge C_g\right).$$

Example 1. We will be using the valid PLTL formula $\Box((a \to b) \to \bigcirc b) \to \Diamond\Box(a \vee b)$ as a running example that will guide us through the whole theorem proving process presented in this paper. By negating the formula and performing standard transformations we obtain $\Box(a \vee \bigcirc b) \wedge \Box(\neg b \vee \bigcirc b) \wedge \Box\Diamond(\neg a \wedge \neg b)$, which gives us the following PLTL-specification $S = (\{a, b\}, \emptyset, \{a \vee b', \neg b \vee b'\}, \{\neg a, \neg b\})$.

It is a known fact that when considering satisfiability of PLTL formulas attention can be restricted to *ultimately periodic* [18] interpretations. These start with a finite sequence of worlds and then repeat another finite sequence of worlds forever. This observation, which is also one of the key ingredients of our approach, motivates the following definition.

Definition 2. *Let $K \in \mathbb{N}$, and $L \in \mathbb{N}^+$ be given. A PLTL-interpretation $(V_i)_{i\in\mathbb{N}}$ is a (K, L)-*model *of $S = (\Sigma, I, T, G)$ if*

1. *for every $C \in I$, $V_0 \models C$,*
2. *for every $i \in \mathbb{N}$ and every $C \in T$, $[V_i, V_{i+1}] \models C$,*
3. *for every $i \in \mathbb{N}$ and every $C \in G$, $V_{(K+i\cdot L)} \models C$.*

A PLTL-specification is satisfiable *if it has a (K, L)-model for some K and L.*

Note that the eventuality represented by the goal clauses of S is satisfied infinitely often as the standard PLTL semantics dictates. Moreover, we keep track of the worlds where this is bound to happen by requiring they form an arithmetic progression with K as the initial term and L the common difference. This additional requirement doesn't change the notion of satisfiability thanks to the observation mentioned above. We will call the pair (K, L) the *rank* of a model.

3 Labelled Clauses

Recall that we defined a PLTL-interpretation as an infinite sequence of propositional valuations over the finite signature Σ. Alternatively though, it can be viewed as a *single* propositional valuation over the *infinite* signature $\Sigma^* = \bigcup_{i\in\mathbb{N}} \Sigma^{(i)}$. We simply index the signature symbols by the time moments to obtain this isomorphic representation. If we now examine Definition 2 of (K, L)-models from this perspective, we can reveal a simple (though at first sight not very useful) reduction of satisfiability in a (K, L)-model to propositional satisfiability of a potentially infinite set of clauses over Σ^*. For a specification $S = (\Sigma, I, T, G)$ this clause set will consist of copies of the clauses from I, T, and G that are "shifted in time" to proper positions, such that the whole set is (propositionally) satisfiable if and only if S has a (K, L)-model. Formally, the set is the union of $\{C^{(0)} \mid C \in I\}$, $\{C^{(i)} \mid C \in T, i \in \mathbb{N}\}$, and $\{C^{(K+i\cdot L)} \mid C \in G, i \in \mathbb{N}\}$. See Fig. 1 for the intuition behind this idea.

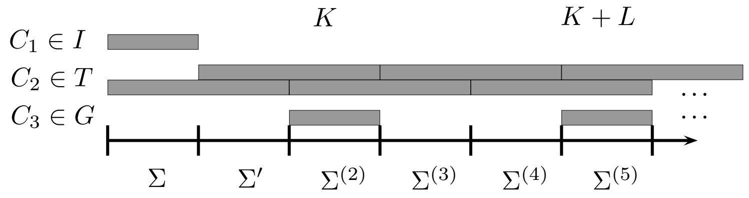

Fig. 1. Schematic presentation of the potentially infinite set of clauses that is satisfiable iff a PLTL-specification $S = (\Sigma, I, T, G)$ has a model of rank $(2, 3)$

In order to make use of the above described reduction we need to show how to solve for infinitely many values of K and L the propositional satisfiability problem consisting of infinitely many clauses. We do this by assigning *labels* to the clauses of S such that a labelled clause represents up to infinitely many standard clauses over Σ^*. Then an inference performed between labelled clauses corresponds to infinitely many inferences on the level of Σ^*. This is not dissimilar to the idea of "lifting" from first-order theorem proving where clauses with variables represent up to infinitely many ground instances. Here, however, we deal with the additional dimension of performing infinitely many proof tasks "on the ground level" in parallel, one for each rank (K, L).

Formally, a *label* is a pair (b, k) where b is either $*$ or 0, and k is either $*$ or an element of $\mathbb{N}$. A *labelled clause* is a pair $(b, k) \,||\, C$ consisting of a label and a (standard) clause over $\Sigma \cup \Sigma'$. Given a PLTL-specification $S = (\Sigma, I, T, G)$, the *initial labelled clause set N_S* for S is defined to contain

- labelled clauses of the form $(0, *) \,||\, C$ for every $C \in I$,
- labelled clauses of the form $(*, *) \,||\, C$ for every $C \in T$, and
- labelled clauses of the form $(*, 0) \,||\, C$ for every $C \in G$.

We can think of the first label component b as relating the clause to the beginning of time, while the second component relates the clause to the indices of the form $K + i \cdot L$, where the goal should be satisfied. In both cases, $*$ stands for a "don't care" value, thus, e.g., the label $(*, *)$ marks clauses that occupy every possible index. It turns out that during inferences we also need to talk about clauses that reside k steps before indices of the goal. That is why the second label component may assume any value from $\mathbb{N}$. The semantics of labels is given via a map to world indices.

Let (K, L) be a rank. We define a set $R_{(K,L)}(b, k)$ of indices *represented* by the label (b, k) as the set of all $t \in \mathbb{N}$ such that

$$[b \neq * \rightarrow t = 0] \wedge [k \neq * \rightarrow \exists s \in \mathbb{N}. t + k = K + s \cdot L] \enspace .$$

Observe that while $R_{(K,L)}(0, k) \subseteq \{0\}$, the sets $R_{(K,L)}(*, k)$ are always infinite, and for $k \in \mathbb{N}$ constitute a range of an arithmetic progression with difference L. Now a standard clause of the form $C^{(t)}$ is said to be *represented by the labelled clause $(b, k) \,||\, C$ in (K, L)* if $t \in R_{(K,L)}(b, k)$. We denote the set of all standard clauses represented in (K, L) by the labelled clauses N by the symbol $N_{(K,L)}$.

$$N_{(K,L)} = \{C^{(t)} \mid \text{clause } (b, k) \,||\, C \in N \text{ and } t \in R_{(K,L)}(b, k)\} \enspace .$$

Example 2. Our example specification $S = (\{a, b\}, \emptyset, \{a \vee b', \neg b \vee b'\}, \{\neg a, \neg b\})$ contains among others the single literal goal clause $\neg a$. In the initial labelled clause set N_S this goal clause becomes $(*, 0) \,||\, \neg a$. If we now, for example, fix the same rank $(2, 3)$ as in Fig. 1, our labelled clause will in that rank represent all the standard clauses $(\neg a)^{(t)}$ with $t \in R_{(2,3)}(*, 0) = \{2, 5, 8, \dots\}$.

We summarize the main message of this section in the next lemma. Its proof follows from the definitions and ideas already given.

Lemma 1. *Let a rank (K, L) and a PLTL-specification S be given and let N_S be the initial labelled clause set for S. Then the set $(N_S)_{(K,L)}$ is satisfiable if and only if S has a (K, L)-model.*

4 The Labelled Superposition Calculus *LPSup*

In this section we present our calculus for labelled clauses *LPSup*. We lift the ordered resolution calculus of [2], which we call *PSup* for Propositional Superposition, and transfer to *LPSup* its valuable properties, including the ordering restrictions of inferences. For that purpose, we parameterize *LPSup* by a total ordering $<$ on the symbols of the signature Σ, which we implicitly extend to indexed signatures by first comparing the indices and only then the actual symbols. This means that $p^{(i)} < q^{(j)}$ if and only if $i < j$, or $i = j$ and $p < q$.[1] We then use the standard extension of this ordering to compare literals in clauses.

Before we proceed to the actual presentation of the calculus, we need to define how labels are updated by inferences. Two labelled clauses should only interact with each other when they actually represent standard clauses that interact on the ground level. Moreover, the resulting labelled clause should represent exactly all the possible results of the interactions on the ground level. We define the *merge* of two labels (b_1, k_1) and (b_2, k_2) as the label (b, k) such that

– if $b_1 = b_2 = *$ then $b = *$, otherwise $b = 0$,
– if $k_1 = *$ then $k = k_2$; if $k_2 = *$ then $k = k_1$; if $k_1 = k_2 \neq *$ then $k = k_1 = k_2$.

In the case when $k_1, k_2 \in \mathbb{N}$ and $k_1 \neq k_2$, the merge operation is *undefined.* The idea is that the merged label represents the intersection of the sets of indices represented by the arguments.

The calculus *LPSup* consists of the inference rules Ordered Resolution, Ordered Factoring, Temporal Shift, and Leap. They operate on a clause set N, an initial labelled clause set of a given PLTL-specification. While Ordered Resolution and Ordered Factoring constitute the labelled analogue of inferences of *PSup*, Temporal Shift and Leap are "structural" in nature, as they only modify the syntactic format, but the underlying represented set of standard clauses remains the same.

(i) *Ordered Resolution*

$$\mathcal{I}\frac{(b_1, k_1)\,||\,C \vee L \quad (b_2, k_2)\,||\,D \vee \bar{L}}{(b, k)\,||\,C \vee D}$$

where literal L is maximal in C, its complement $\bar{L}$ is maximal in D, and the merge of labels (b_1, k_1) and (b_2, k_2) is defined and equal to (b, k),

(ii) *Ordered Factoring*

$$\mathcal{I}\frac{(b, k)\,||\,C \vee A \vee A}{(b, k)\,||\,C \vee A}$$

where A is an atom maximal in C,

[1] In the case of labelled clauses this amounts to saying that the symbols of Σ' are considered larger than those of Σ. Our definition, however, also makes sense over the infinite signature Σ^* and it is this particular ordering that restricts the inferences on the level of standard clauses.

(iii) *Temporal Shift*

$$\mathcal{I}\frac{(*,k)\,||\,C}{(*,k')\,||\,(C)'}$$

where C is a clause over Σ only, and $k = k' = *$ or $k \in \mathbb{N}$ and $k' = k+1$,

(iv) *Leap*

$$\mathcal{I}\frac{\{(b,u+i\cdot v)\,||\,C\}_{i\in\mathbb{N}} \text{ derivable from } N}{(b,u-v)\,||\,C}$$

where $u \geq v > 0$ are integers and C is an arbitrary standard clause.

Further explanation is needed for the inference rule Leap. In its present form it requires an infinite number of premises, one for each $i \in \mathbb{N}$, and thus cannot, strictly speaking, become applicable in any finite derivation. Here it is only a mathematical abstraction. In the next section we show how to effectively generate and finitely represent infinite sets of labelled clauses from which it will follow that Leap is, in fact, effective.

Going back to the other inferences note that the merge operation on labels ensures that the conclusion of Ordered Resolution represents exactly all the conclusions of the standard ordered resolution inferences between the standard clauses represented by the premises. Ordered Factoring carries over from *PSup* in a similar fashion.[2] The Temporal Shift operates only on clauses over the signature Σ. We will from now on call such clauses *simple*. Notice that the restriction to simple clauses is essential as it keeps the symbols of the conclusion to stay within $\Sigma \cup \Sigma'$.

Example 3. The initial labelled clause set N_S of our running example contains among others also clauses $(*,*)\,||\,a \vee b'$ and $(*,0)\,||\,\neg b$. We can apply Temporal Shift to the second to obtain $(*,1)\,||\,\neg b'$. Now b' is the only literal over Σ' in the first clauses and therefore maximal. So the first clause and the newly derived one can participate in Ordered Resolution inference with conclusion $(*,1)\,||\,a$.

Although the rules Temporal Shift and Leap derive new labelled clauses, the represented sets of standard clauses remain the same in any rank (K,L). This is easy to see for Temporal Shift, but a little bit more involved for Leap, where it relies on the periodicity of (K,L)-models. The overall soundness of *LPSup* is established by relating it to the same property of the standard calculus *PSup*.

Theorem 1 (Soundness of *LPSup*). *Let N_S be the initial labelled clause set for a PLTL-specification S, and $(b,k)\,||\,C$ a labelled clause derivable from N_S by LPSup. Then for any rank (K,L) and any $t \in R_{(K,L)}(b,k)$ the standard clause $C^{(t)}$ is derivable from $(N_S)_{(K,L)}$ by PSup.*

If an empty labelled clause $(b,k)\,||\,\bot$ is derivable from N_S by LPSup, such that $R_{(K,L)}(b,k) \neq \emptyset$, then S doesn't have a (K,L)-model.

[2] Here we present the rule in a form as close as possible to the one in [2]. In practical implementation, however, it is reasonable to remove duplicate literals as soon as they occur without regard to ordering restrictions.

Notice that in *LPSup* the fact that an empty labelled clause $(b, k) \,||\, \bot$ is derived does not necessarily mean that the whole clause set is unsatisfiable. It only rules out those (K, L)-models for which $R_{(K,L)}(b, k)$ is non-empty. This motivates the following definition.

Definition 3. *An empty labelled clause* $(b, k) \,||\, \bot$ *is called* conditional *if* $b = 0$ *and* $k \in \mathbb{N}$*, and* unconditional *otherwise. We say that a set of labelled clauses* N *is* contradictory *if it contains an unconditional empty clause, or* $(0, k) \,||\, \bot$ *is in* N *for every* $k \in \mathbb{N}$*.*

In Sect. 6 we demonstrate that a (K, L)-model can be found for any non-contradictory set of labelled clauses that is saturated by *LPSup*.

To complete the picture of *LPSup* we move on to mention reduction rules. As we discuss in detail in Sect. 6, these are justified by the abstract redundancy notion [2] which our calculus inherits from *PSup*. Thus the following are only examples and other reductions can be developed and used as long as they satisfy the criteria of abstract redundancy.

Tautology Deletion allows us to remove from the search any labelled clause the standard part of which contains both a literal and its complement. Another useful reduction is *Subsumption*[3]

$$\mathcal{R}\frac{(b_1, k_1) \,||\, C \quad (b_2, k_2) \,||\, D}{(b_1, k_1) \,||\, C}$$

where C is a sub-multiset of D and the merge of labels (b_1, k_1) and (b_2, k_2) is defined and equal to (b_2, k_2).

5 Decision Procedure

In this section we explain how to turn the calculus *LPSup* into an effective decision procedure for PLTL. First, we have a look at termination.

Example 4. We have already derived the labelled clause $(*, 1) \,||\, \neg b'$ from our set N_S of initial clauses for S by Temporal Shift. Ordered Resolution between this clause and the clause $(*, *) \,||\, \neg b \vee b'$ yields $(*, 1) \,||\, \neg b$ to which Temporal Shift is again applicable, giving us $(*, 2) \,||\, \neg b'$. We see that the clause we started with differs from the last one only in the label where the k-component got increased by one. The whole sequence of inferences can now be repeated, allowing us to eventually derive labelled clauses $(*, k) \,||\, \neg b$ and $(*, k) \,||\, \neg b'$ for any $k \in \mathbb{N}^+$.

The example demonstrates how the Temporal Shift inference may cause non-termination when the k-component of the generated labelled clauses increases one by one. It also suggests, however, that from a certain point the derived clauses don't add any new information and the inferences essentially repeat in

[3] We use the letter $\mathcal{I}$ and $\mathcal{R}$ to distinguish between *inference rules*, whose premises are kept after the conclusion has been added to the given set of clauses, and *reduction rules*, whose premises are replaced by the conclusion.

cycles. Detecting these repetitions and finitely representing the resulting infinite clause sets is the key idea for obtaining a termination result for our calculus.

Given a set of labelled clauses N, it is convenient to think of N as being separated into *layers*, sets of clauses with the same value of their labels' second component k. This way we obtain the $*$-layer of clauses with the label of the form $(b, *)$ for $b \in \{*, 0\}$, and similarly layers indexed by $k \in \mathbb{N}$. The following list of observations forms the basis of our strategy for saturating clause sets by *LPSup*.

(1) In an initial labelled clause set only the $*$-layer and 0-layer are non-empty.
(2) If all premises of Ordered Resolution, Factoring or Temporal Shift inference belong to the $*$-layer, so does the conclusion of the respective inference.
(3) If a premise of Ordered Resolution or Factoring inference belongs to the k-layer for $k \in \mathbb{N}$, so does the inference's conclusion.
(4) If a premise of Temporal Shift belongs to the k-layer for $k \in \mathbb{N}$, the inference's conclusion belongs to the layer with index $(k + 1)$.
(5) The number of clauses in each layer is bounded by a constant depending only on the size of the signature.

We are ready to describe what we call *layer-by-layer* saturation of an initial labelled clause set. During this process we don't yet consider the Leap inference, which will be incorporated later. It follows from our observations that the $*$-layer can always be finitely saturated. We then perform all the remaining Ordered Resolution and Factoring inferences (together with possible reductions) to saturate the 0-layer, again in a finite number of steps. After that we exhaustively apply the Temporal Shift rule to populate the 1-layer and again saturate this layer by Ordered Resolution and Factoring. This process can be repeated in the described fashion to saturate layers of increasing indices. It is important that the new clauses of the higher layers can never influence (by participating on inferences or reductions) clauses in the lower, already saturated, layers. Eventually, thanks to point (5) above, we will encounter a layer we have seen before and then we stop. More precisely, in a finite number of steps we are bound to obtain a set of labelled clauses N such that there are integers $o \in \mathbb{N}$ and $p \in \mathbb{N}^+$ and

– the o-layer of N is equal to the $(o + p)$-layer of N (up to reindexing[4]),
– the clause set is saturated by *LPSup* (without Leap), except, possibly, for Temporal Shift inferences with premise in layer $(o + p)$,
– the layers with index larger than $(o + p)$ are empty.

Now we need a final observation to finish the argument. The applicability of Ordered Resolution, Factoring and Temporal Shift (as well as that of the reductions of *LPSup*) is "invariant under the move from one layer to another". In other words, exactly the same (up to reindexing) inferences (and reductions) that have been performed to obtain, e.g., the saturated layer of index $(o + 1)$, can now be repeated to obtain the saturated layer of index $(o + p + 1)$. We can therefore stop the saturation process here and define:

[4] Meaning the first mentioned set would be identical to the second if we changed the second label component of all its clauses from o to $(o + p)$.

Definition 4. *Let N be a clause set obtained by layer-by-layer saturation as described above. We call the numbers o and p the* offset *and* period *of N, respectively. The* infinite extension *of such N is the only set of labelled clauses N^* for which $N \subseteq N^*$ and such that for every $i \in \mathbb{N}$ the $(o+i)$-layer of N^* is equal to the $(o + i \bmod p)$-layer of N (up to reindexing).*

The infinite extension of N is completely saturated by *LPSup* (without Leap).

Example 5. In our running example, the $*$-layer and 0-layer are already saturated. The next layers we obtain are

$$\{(*,1) \,||\, \neg a', (*,1) \,||\, \neg b', (*,1) \,||\, a, (*,1) \,||\, \neg b\} \ , \tag{1}$$

$$\{(*,2) \,||\, a', (*,2) \,||\, \neg b', (*,2) \,||\, a, (*,2) \,||\, \neg b\} \tag{2}$$

As the 3-layer is then equal to the previous (up to reindexing), layer-by-layer saturation terminates with offset 2 and period 1.

In layer-by-layer saturation we always give priority to Ordered Resolution and Factoring inferences, and only when these are no longer applicable in the current clause set, we perform all the pending Temporal Shift inferences, and possibly repeat. Similarly, the *overall saturation procedure* which we present next combines layer-by-layer saturation phases with an exhaustive application of the Leap inference:

1. Set N_1 to the initial labelled clause set N_S of a given PLTL-specification S.
2. Set N_2 to the layer-by-layer saturation on N_1.
3. If the clause set N_2^* is contradictory, stop and report UNSAT.
4. Set N_3 to be the set N_2 enriched by all the possible conclusions of Leap inference with premises in N_2^*, possibly reduced.
5. If $N_3 = N_2$ stop and report SAT, else go back to step 2 resetting $N_1 := N_3$.

Note that if we go to line 2 for the second time, N_1 is no longer an initial labelled clause set. Although we didn't discuss it previously, it is straightforward to perform layer-by-layer saturation of any finitely represented clause set.

On lines 3 and 4 we refer to the infinite extension N_2^*. It actually means that we operate with the layer-by-layer saturation N_2 together with offset o and period p. Now N_2^* is bound to be contradictory if and only if N_2 contains an unconditional empty clause or $(0,k) \,||\, \bot$ is in N_2 for every $0 \leq k < o + p$. Similarly, a labelled clause $(b, j) \,||\, C$ with $j < o$[5] can be derived by Leap inference with premises in N_2^* if and only if there is a clause $(b,i) \,||\, C$ in N_2 such that $o \leq i < o + p$ and p divides $i - j$.

Finally note that while the values of offset and period associated with N_2 may change from one repetition to another, their sum is each time bounded by the same constant depending only on the size of the signature, namely the number of different possible layers (up to reindexing). Moreover, thanks to the fact that we only work with a fixed finite signature, there is also a bound on the number of non-trivial additions to the individual layers on line 4. These together ensure that the procedure always terminates.

[5] Leap conclusion with $j \geq o$ is always redundant.

Example 6. In our example, the infinite extension of the layer-by-layer saturation contains the premises $\{(*, 1 + i) \,||\, a\}_{i\in\mathbb{N}}$ of a Leap inference with conclusion $(*, 0) \,||\, a$. This clause together with the already present $(*, 0) \,||\, \neg a$ gives us the empty clause $(*, 0) \,||\, \perp$ by Ordered Resolution, which eventually terminates the overall procedure, because the empty clause is unconditional and therefore the overall set becomes contradictory.

6 Redundancy, Completeness and Model Building

The calculus *LPSup* comes with an abstract notion of redundancy in the spirit of [2]. Also here one can recognize the idea of "lifting", which relates the standard level of *PSup* to the level of labelled clauses. Recall that a standard clause C is called redundant with respect to a set of standard clauses N if there are clauses $C_1, \dots, C_n \in N$ such that for every $i = 1, \dots n$, $C_i < C$, and $C_1, \dots, C_n \models C$. On the level of labelled clause we define:

Definition 5. *A labelled clause* $(b, k) \,||\, C$ *is* redundant *with respect to a set of labelled clauses* N*, if for any rank* (K, L) *every standard clause represented by* $(b, k) \,||\, C$ *in* (K, L) *is redundant w.r.t.* $N_{(K,L)}$.

A set of labelled clauses N *is* saturated up to redundancy *with respect to LPSup, if for every inference from* N *such that its premises are not redundant w.r.t.* N*, the conclusion is either redundant w.r.t.* N *or contained in* N.

Note that the reductions of *LPSup* described in Sect. 4 are instances of redundancy elimination. This is easy to see for Tautology deletion, and follows from the semantics of the merge operation on labels for the Subsumption reduction. It is important to note that these are just examples and further reductions can be developed and used. As long as they fit into the framework prescribed by Definition 5, they are guaranteed to preserve completeness and the underlying proof need not be changed.

Our main theorem relates completeness of *LPSup* to the same property of the underlying calculus *PSup* via the notion of redundancy.

Theorem 2 (Completeness of ***LPSup*****).** *Let* N *be a labelled clause set saturated in a layer-by-layer fashion with offset* o *and period* p *and let* N^**, the infinite extension of* N*, be a non-contradictory set of labelled clause saturated up to redundancy w.r.t. LPSup. We set* K *to be the smallest number from* $\mathbb{N}$ *such that* $(0, K) \,||\, \perp$ *is not in* N^* *(note that* N^* *is non-contradictory), and further set* L *to the smallest positive multiple of* p *that is not smaller than* o*. Then the set* $N^*_{(K,L)}$ *does not contain the (standard) empty clause and is saturated up to redundancy w.r.t. PSup.*

Recall the overall saturation procedure of the previous section. Its input is a PLTL-specification which is immediately transformed into the initial labelled clause set. If the procedure reports UNSAT, we know the input is unsatisfiable, because we derived (using a sound calculus) a contradictory set of labelled clauses, which rules out any (K, L)-model. If, on the other hand, the procedure reports SAT, we may apply Theorem 2 together with completeness of *PSup* to conclude that the set $N^*_{(K,L)}$ is satisfiable, and, therefore, the specification we started with has a (K, L)-model. Thus the overall saturation procedure decides satisfiability of PLTL-specifications.

We close this section by commenting on the possibility of using our method to provide counterexamples to non-valid PLTL formulas. Due to space restrictions, we cannot describe the method in full detail, but to those familiar with the model construction for classical logic based on *PSup* [2], it should be clear that with Theorem 2 proven, we are practically done.

Given a non-contradictory set of labelled clauses N^* that is saturated up to redundancy w.r.t. *LPSup*, we pick (K, L) as described in Theorem 2 and generate the standard clauses of $N^*_{(K,L)}$ one by one with increasing $<$. We apply classical model construction to these clauses to gradually build a (partial) valuation over $\Sigma^* = \bigcup_{i\in\mathbb{N}} \Sigma^{(i)}$, which, as we know, corresponds in the obvious way to a (K, L)-model $(V_i)_{i\in\mathbb{N}}$. We can stop the generation as soon as a particular (already completed) valuation repeats (i.e. $V_i = V_{i-j}$ for some $j \in \mathbb{N}^+$) and the goal has already been reached (i.e. $i > K$). An ultimately periodic model is then output as a result.

7 Final Discussion and Experiments

We now compare our calculus to Clausal Temporal Resolution [6]. Older resolution based approaches to PLTL are [3, 21], but they don't seem to be used or developed any further nowadays. Besides resolution there are approaches to PLTL satisfiability based on tableaux deduction [22, 17], and on automata theory [16]. These seem to be less related and we don't discuss them here further.

It can be shown that operationally there is a close connection between *LPSup* and the Clausal Temporal Resolution (CTR) of [6]. From this perspective, our formalism of labelled clauses can be seen as a new way to derive completeness of CTR that justifies the use of ordering restrictions and redundancy elimination. This has not been achieved yet in full by previous work: [9] contains a proof theoretic argument, but only for the use of ordering restrictions, [11] sketches the idea how to justify tautology removal and subsumption, but not the general redundancy notion in the style of [2] that we provide.

Moreover, there is also a correspondence between our layer-by-layer saturation followed by the application of the Leap inference and the BFS-Loop search of CTR as described in [7, 13]. Apart from being interesting in its own right, this view sheds new light on explaining BFS-Loop search, as it gives meaning to the intermediate clauses generated in the process, and we thus don't need to take the detour through the DNF representation of [5]. Even here, the idea of labels clearly separates logical content of the clauses from the meta-logical one (c.f. the ad hoc marker literal of [7]).

Despite these similarities between *LPSup* and CTR, the calculi are by no means identical. As discussed before, a temporal model can be extracted in a straightforward way from a satisfiable set of labelled clauses saturated by *LPSup*.

This doesn't hold for CTR, where a more complex approach that simulates the model construction of [2] only locally needs to be applied [14]. In particular, because saturation by CTR doesn't give the model building procedure any guidance as to where to look for the goal, in each considered world all the possible orderings on the signature (in the worse case) need to be tried out in a fair way to make sure a goal world is eventually reached. As each change of the ordering calls for a subsequent resaturation of the clause set in question (so that the local model construction still works), it obviously diminishes the positive effect orderings in general have on reducing the search space.

Finally note that since we eventually rely on propositional superposition, we can also take into account the explicit use of partial models to further guide the search for a proof or saturation. The idea is to build a partial model based on the ordering on propositional literals. Then it can be shown that resolution can be restricted to premises where one is false and the other true in the partial model [1]. This superposition approach on propositional clauses is closely related to the state of the art CDCL calculus (see, e.g. [23]) for propositional logic. The missing bit is to "lift" this setting to our labelled clauses. This will be one direction for future research.

We implemented a simple prototype of both *LPSup* and CTR (with BFS loop-search in the style of [7]), in order to compare the two calculi on non-trivial examples. In this section we briefly report on our experiment. The prototype, written in SWI-Prolog, is available along with the test examples at [19].

For the experiment we choose two formula families described in [10], which we call $\mathcal{C}_n^1$ and $\mathcal{C}_n^2$. In addition, we also tested the calculi on formulas from two families specifically constructed to highlight the respective weaknesses of *LPSup* and CTR. These we call the implicit and explicit cycles problems, respectively, and denote them by $\mathcal{I}_{(l_1+\dots+l_k)}$ and $\mathcal{E}_{(l_1+\dots+l_k)}$. The problems are parameterized by the sequence of numbers $l_1, \dots, l_k$, which denote the cycles' lengths.

Table 1. Results of comparing our implementations of *LPSup* and CTR with TRP++

		LPSup-Prolog			CTR-Prolog			TRP++	
Problem	Size	Cl-gen	Lits-gen	Cl-subs	Cl-gen	Lits-gen	Cl-subs	Cl-gen	Cl-subs
$\mathcal{C}_{10}^1$	56	53	202	100	174	576	110	363	300
$\mathcal{C}_{15}^1$	81	78	377	145	334	1161	240	688	595
$\mathcal{C}_{20}^1$	106	103	602	190	544	1946	420	1113	990
$\mathcal{C}_3^2$	22	442	1376	324	984	3972	909	1146	968
$\mathcal{C}_4^2$	30	1937	7649	1612	5298	26086	5047	3560	3053
$\mathcal{C}_5^2$	38	6287	28576	5635	18724	102704	18134	7925	6922
$\mathcal{I}_{(3+5)}$	62	406	1563	368	203	1022	194	86	160
$\mathcal{I}_{(3+5+8)}$	253	8010	42024	7356	1087	7613	1145	390	745
$\mathcal{E}_{(2+3)}$	8	23	25	4	131	424	78	177	77
$\mathcal{E}_{(2+3+4)}$	13	52	55	6	1061	4490	595	1597	627

Table 7 summarizes the results of our experiments. For each problem and for both calculi we report the number of clauses in the input, the number of derived[6] clauses and literals, and the number of subsumed clauses. For comparison, we also include in the last two columns clause data obtained by running the temporal prover TRP++ [8][7], which also implements the CTR calculus, to provide evidence that our experimental results are not biased. We decided not to report on running times as our aim here is to compare the calculi rather than the implementations. The number of generated clauses (literals) should provide a good measure on the amount of data to be processed by any prover, which is, moreover, independent on the choice programming language or the use of particular data structures.

As we can see, *LPSup* needs to generate consistently less clauses to draw its conclusion for both $\mathcal{C}_n^1$ and $\mathcal{C}_n^2$. It only behaves worse on the implicit cycles examples $\mathcal{I}$, which are constructed in such a way that the number of iterations of the layer-by-layer saturation is much higher for *LPSup* than CTR. The examples $\mathcal{E}$, on the other hand, present much more work for CTR, where the equivalent of Temporal Shift rule causes the clause set to "blow-up". All in all, *LPSup* seems to come considerably better off out of our experiments.

8 Conclusion

We applied the ideas of labelled superposition to develop a new decision procedure for propositional linear temporal logic. On the presentation level, it replaces the complex temporal resolution rule from the previously proposed calculus by a simple check for repetition in the derived clause set and a subsequent inference. Its unique treatment of goal clauses enables straightforward partial model building of satisfiable clause sets which could potentially be used to further restrict inferences. Moreover, the experimental comparison to previous work suggests that the new calculus typically explores smaller search spaces to derive its conclusion. Development of an optimized implementation, to be tested on a set of representative benchmarks, will be part of our future work.

References

[1] Bachmair, L., Ganzinger, H.: On Restrictions of Ordered Paramodulation with Simplification. In: Stickel, M.E. (ed.) CADE 1990. LNCS, vol. 449, pp. 427–441. Springer, Heidelberg (1990)

[2] Bachmair, L., Ganzinger, H.: Resolution theorem proving. In: Robinson, J.A., Voronkov, A. (eds.) Handbook of Automated Reasoning, pp. 19–99. Elsevier and MIT Press (2001)

[6] This covers all the resolvents, plus the clauses derived by non-trivial Leap inference. (Leap conclusions subsumed by other clauses are not generated at all.)

[7] We used version 2.1 available at `http://www.csc.liv.ac.uk/~konev/software/trp++/`.

[3] Cavalli, A., del Cerro, L.: A Decision Method for Linear Temporal Logic. In: Shostak, R.E. (ed.) CADE 1984. LNCS, vol. 170, pp. 113–127. Springer, Heidelberg (1984)

[4] Degtyarev, A., Fisher, M., Konev, B.: A Simplified Clausal Resolution Procedure for Propositional Linear-Time Temporal Logic. In: Egly, U., Fermüller, C. (eds.) TABLEAUX 2002. LNCS (LNAI), vol. 2381, pp. 85–99. Springer, Heidelberg (2002)

[5] Dixon, C.: Search Strategies for Resolution in Temporal Logics. In: McRobbie, M.A., Slaney, J.K. (eds.) CADE 1996. LNCS, vol. 1104, pp. 673–687. Springer, Heidelberg (1996)

[6] Fisher, M., Dixon, C., Peim, M.: Clausal temporal resolution. ACM Trans. Comput. Logic 2, 12–56 (2001)

[7] Fernández Gago, M.C., Fisher, M., Dixon, C.: Algorithms for Guiding Clausal Temporal Resolution. In: Jarke, M., Koehler, J., Lakemeyer, G. (eds.) KI 2002. LNCS (LNAI), vol. 2479, pp. 235–252. Springer, Heidelberg (2002)

[8] Hustadt, U., Konev, B.: TRP++ 2.0: A Temporal Resolution Prover. In: Baader, F. (ed.) CADE-19. LNCS (LNAI), vol. 2741, pp. 274–278. Springer, Heidelberg (2003)

[9] Hustadt, U., Konev, B., Schmidt, R.A.: Deciding Monodic Fragments by Temporal Resolution. In: Nieuwenhuis, R. (ed.) CADE 2005. LNCS (LNAI), vol. 3632, pp. 204–218. Springer, Heidelberg (2005)

[10] Hustadt, U., Schmidt, R.: Scientific benchmarking with temporal logic decision procedures. In: KR 2002, pp. 533–546. Morgan Kaufmann (2002)

[11] Konev, B., Degtyarev, A., Dixon, C., Fisher, M., Hustadt, U.: Mechanising first-order temporal resolution. Inf. Comput. 199, 55–86 (2005)

[12] Lev-Ami, T., Weidenbach, C., Reps, T., Sagiv, M.: Labelled Clauses. In: Pfenning, F. (ed.) CADE 2007. LNCS (LNAI), vol. 4603, pp. 311–327. Springer, Heidelberg (2007)

[13] Ludwig, M., Hustadt, U.: Fair Derivations in Monodic Temporal Reasoning. In: Schmidt, R.A. (ed.) CADE-22. LNCS, vol. 5663, pp. 261–276. Springer, Heidelberg (2009)

[14] Ludwig, M., Hustadt, U.: Resolution-based model construction for PLTL. In: TIME 2009, pp. 73–80. IEEE Computer Society (2009)

[15] Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science, pp. 46–57. IEEE (1977)

[16] Rozier, K.Y., Vardi, M.Y.: LTL Satisfiability Checking. In: Bošnački, D., Edelkamp, S. (eds.) SPIN 2007. LNCS, vol. 4595, pp. 149–167. Springer, Heidelberg (2007)

[17] Schwendimann, S.: A New One-Pass Tableau Calculus for PLTL. In: de Swart, H. (ed.) TABLEAUX 1998. LNCS (LNAI), vol. 1397, pp. 277–291. Springer, Heidelberg (1998)

[18] Sistla, A.P., Clarke, E.M.: The complexity of propositional linear temporal logics. J. ACM 32, 733–749 (1985)

[19] Suda, M., Weidenbach, C.: Prototype implementation of LPSup. (2011), `http://www.mpi-inf.mpg.de/~suda/supLTL.html`

[20] Suda, M., Weidenbach, C.: Labelled Superposition for PLTL. Research Report MPI-I-2012-RG1-001, Max-Planck-Institut für Informatik, Saarbrücken (2012)

[21] Venkatesh, G.: A Decision Method for Temporal Logic Based on Resolution. In: Maheshwari, S.N. (ed.) FSTTCS 1985. LNCS, vol. 206, pp. 272–289. Springer, Heidelberg (1985)

[22] Wolper, P.: The tableau method for temporal logic: An overview. Logique et Analyse 28, 119–136 (1985)

[23] Zhang, L., Madigan, C.F., Moskewicz, M.W., Malik, S.: Efficient conflict driven learning in boolean satisfiability solver. In: ICCAD, pp. 279–285 (2001)