Linear Temporal Logic and Propositional Schemata, Back and Forth

Vincent Aravantinos, Ricardo Caferra, Nicolas Peltier

Linear Temporal Logic and Propositional Schemata, Back and Forth*

Vincent Aravantinos, Ricardo Caferra, Nicolas Peltier
Laboratory of Informatics of Grenoble (CNRS, Grenoble INP)
Bâtiment IMAG C - 220 rue de la Chimie 38400 Saint Martin d'Hères
{vincent.aravantinos,ricardo.caferra,nicolas.peltier}@imag.fr

Abstract

This paper relates the well-known Linear Temporal Logic [22] with the logic of propositional schemata *introduced in [1]. We prove that LTL is equivalent to a class of schemata in the sense that polynomial-time reductions exist from one logic to the other. Some consequences about complexity are given. We report about first experiments and the consequences about possible improvements in existing implementations are analyzed.*

1. Introduction

Linear Temporal Logic (LTL) is a very well-known logic introduced in [22] for verifying computer programs. It is widely used to reason on finite state transition systems. On the other hand, *propositional schemata* have been introduced in [1]. They extend the language of propositional logic with *indexed propositions* (such as p_{n}, p_1 or $p_{\mathsf{i}+1}$) and *iterated connectives* of the form $\bigvee_{\mathsf{i}=0}^{\mathsf{n}} \phi$ or $\bigwedge_{\mathsf{i}=0}^{\mathsf{n}} \phi$. Note that n denotes a parameter, interpreted as a natural number. If arbitrary expressions for indices and iterations are allowed in the schema, then the satisfiability problem is undecidable, but we have identified in [1, 2, 4] some subclasses for which this problem is decidable. The simplest of these classes is called *regular*: it is defined by restricting both the indices of the propositions, that must be of the form k or $\mathsf{n}+k$ where $k \in \mathbb{Z}$ and n is a variable, and iterations, that must be non-nested and of the form $\bigwedge_{\mathsf{i}=k}^{\mathsf{n}+l} \phi$ where n is a variable and $k, l \in \mathbb{Z}$. Decision procedures are designed in [1, 2] and an implementation is available [3].

LTL and propositional schemata share many common features and trying to compare them precisely is a rather natural and, hopefully, fruitful idea. In both logics, interpretations can be viewed as arrays of propositional functions and the formulae relate the values of these functions at different states. The *indices* of the propositions in the schematic case may be viewed as the *time* in LTL. Thus comparing the expressing powers and complexities of those two logics, and, if possible, defining translations from one logic to the other is a natural and potentially rewarding issue. Notice that there already exist several results relating LTL to other formalisms like monadic second order logic via Büchi automata [29], monadic first order logic over natural numbers [16] or star-free regular languages [27]. However, there is a fundamental difference between these languages and the logic of schemata: they deal with infinite objects (infinite interpretations in the case of LTL or first order logic over natural numbers, infinite words in the case of star-free regular languages), whereas schemata deal with intrinsically finite (but unbounded) interpretations. This subtle but important difference introduces difficulties in the definition of such translations. This topic bears some similarities with the approach of [10] where problems on Bücchi automata are reduced to problems on finite automata by using the ultimately periodic property of ω-regular languages.

Note that finite interpretation is sometimes a desired feature: restricting LTL to finite traces has been considered in [14], and has applications in, e.g., planning or runtime verification [5, 6, 8]. It can be argued that the use of LTL in such contexts is a bit overkilling. Indeed, often, rather than considering finite traces per se, the preferred approach is to turn them into infinite traces by infinitely repeating the last state. It seems to us that it would be more natural to use schemata for such applications. In the present work, it is shown that doing so entails no loss in expressive power.

In the present paper, we show that LTL is equivalent to a particular subclass of regular schemata, referred to as *sequential*. More precisely, we define algorithms translating formulae from one logic into the other and preserving satisfiability. We believe that these results are interesting from a theoretical perspective since they provide useful information about the expressive power of the respective formalisms. Furthermore they allow to import the complexity results of LTL into schemata. From a practical point of view, the existence of a polynomial reduction from a class of propositional schemata into LTL allows one to ben-

*This work has been partly funded by the project ASAP of the French *Agence Nationale de la Recherche* (ANR-09-BLAN-04-07-01)

efit from the many existing efficient decision procedures for this logic (tableaux methods, e.g. [28, 24], resolution-based methods, e.g. [15], or reductions to model checking, e.g. [23, 13]), implementations [7, 20, 11, 13] and experimentation tools [17]. Conversely, the reverse reduction might give further ideas for the design of new techniques to decide LTL satisfiability. In particular, since a DPLL-based procedure exists for schemata [2], it might help to design such a procedure for LTL. On another hand, this reduction is very reminiscent of the translation from LTL to propositional logic encountered in bounded model checking (BMC) [9]. Contrarily to BMC however, our reduction is *complete*, it might thus give new ideas to achieve completeness in BMC.

The paper is structured as follows. In Section 2 we define LTL and the logic of propositional schemata. In Section 3 we show how to relate the interpretations of both formalisms. A polynomial algorithm transforming any sequential schema into an equivalent LTL formula is presented in Section 4, and Section 5 tackles the reverse translation, i.e. from LTL formulae to schemata. Section 6 presents the results about first experiments with those translations and sketches the possible improvements inspired by those experiments. Finally, Section 7 briefly concludes our work.

The extended version of the present paper (`http://membres-liglab.imag.fr/aravantinos/Site/Publications_files/LTLextended/doc.pdf`) contains additional figures, examples and *all proofs that are omitted here due to space restrictions*.

2. Definitions and notations

In the following, ϕ, ϕ_1, ϕ_2 denote LTL formulae, s, s_1, s_2 denote schemata, σ denotes an LTL or propositional interpretation, $\mathfrak{I}, \mathfrak{M}$ denote schema interpretations, e, f, g denote (Presburger) arithmetic expressions, n, i denote arithmetic variables (n will be used for a free arithmetic variable and i for a bound one). Note that n, i are written in sans serif in order to distinguish them from meta variables denoting natural numbers, that will be written n, i.

Both LTL and schemata have propositional logic as a common basis. Furthermore, in both languages, propositional variables are accompanied with a natural number (an instant in the case of LTL, an index for schemata). So instead of defining, as in classical propositional logic, an interpretation as a function mapping each propositional variable to a truth value, we rather define interpretations as functions mapping *pairs* of propositional variables *and natural numbers* to truth values. Formally:a *propositional interpretation* over a set of *propositional variables* $\mathcal{P}$ is a function from $\mathcal{P} \times \mathbb{N}$ to $\{\mathbf{T}, \mathbf{F}\}$. An interpretation σ is represented by the set of all pairs (variable, natural number) that are true in σ. Most of the time we do not need to make that set explicit. For instance, when interpreting a given formula ϕ, it will be implicitly assumed that we consider only interpretations over sets that contain the variables of ϕ.

2.1. LTL

The syntax of LTL formulae over the set of propositional variables $\mathcal{P}$ is given by the following grammar:

$$\phi ::= \top \mid \mathcal{P} \mid \neg\phi \mid \phi \wedge \phi \mid \mathrm{X}\phi \mid \phi \mathrm{U} \phi$$

$\mathrm{X}\phi$ means that ϕ holds at the next instant ("X" for neXt). $\phi \mathrm{U} \psi$ means that ϕ holds until ψ holds ("U" for Until). We will also use the following abbreviations: $\mathrm{F}\phi \stackrel{\text{def}}{=} \top \mathrm{U} \phi$ and $\mathrm{G}\phi \stackrel{\text{def}}{=} \neg \mathrm{F} \neg \phi$, meaning respectively "$\phi$ eventually holds" and "ϕ always holds". The abbreviations $\vee$, $\Rightarrow$ and $\Leftrightarrow$ are defined as usual (the naive elimination of $\Leftrightarrow$ is exponential but it can be made linear by using renaming of subformulae as usual, which preserves satisfiability). See [22] for details.

LTL formulae are usually interpreted over infinite paths in a transition system, together with a labelling that maps every state to a set of propositional variables. Such sequences are often called *computations* or *behaviours*. We will simply call them *LTL interpretations*. For uniformity, we define formally an LTL interpretation as a propositional interpretation in the sense given above (we do not make explicit the notions of states, transition systems and labelling). Then $\sigma(t)$ denotes the set of variables p that are true at time t, i.e. such that $(p, t) \in \sigma$. The satisfaction relation of an LTL formula ϕ under such an interpretation σ is defined w.r.t. an instant t, written $\sigma, t \models \phi$. This means that the formula ϕ holds at time t. Formally: let ϕ be an LTL formula, σ be a propositional interpretation and $t \in \mathbb{N}$. The relation $\sigma, t \models \phi$ is inductively defined as follows: $\sigma, t \models \top$, $\sigma, t \models p$ iff $(p, t) \in \sigma$, $\sigma, t \models \neg\phi$ iff $\sigma, t \not\models \phi$, $\sigma, t \models \phi_1 \wedge \phi_2$ iff $\sigma, t \models \phi_1$ and $\sigma, t \models \phi_2$, $\sigma, t \models \mathrm{X}\phi$ iff $\sigma, t+1 \models \phi$ and $\sigma, t \models \phi_1 \mathrm{U} \phi_2$ iff $\exists k \in \mathbb{N}$ s.t. $\forall i \in \mathbb{N}, i < k \Rightarrow \sigma, t+i \models \phi_1$ and $\sigma, t+k \models \phi_2$. The notation $\sigma \models \phi$ means that ϕ is true in σ at time 0.

A fundamental property of LTL is the "ultimately periodic model property" [26]. Namely, if an LTL formula is satisfiable, then it is satisfiable on some ultimately periodic interpretation. An *ultimately periodic ("UP") interpretation* is an LTL interpretation σ s.t. there exist $k, l \in \mathbb{N}$ s.t. $l > 0$ and for all $m \geq k$, $\sigma(m) = \sigma(m+l)$. The sequence $\sigma(0) \ldots \sigma(k-1)$ is the *prefix* of σ and $\sigma(k) \ldots \sigma(k+l-1)$ its *loop*, k is the *prefix index* and l is the *period*. The UP model property allows to focus on *finite* sets of instants only, since a UP model is uniquely characterized by the natural numbers k, l and the truth value of propositional variables between time 0 and $k+l$.

2.2. Schemata

We now present the syntax and semantics of schemata. Since the present paper focusses only on a special kind of schemata, called "sequential", we only define this subclass (we refer to, e.g., [4] for the general definition). Let $\mathcal{P}$ be a set of *propositional variables*. Let n, i be two distinct symbols called *arithmetic variables*. We call n the *parameter* and i the *index variable*. For every $p \in \mathcal{P}$, $k \in \mathbb{N}$[1]: p_k, $p_{\mathsf{n}+k}$ and $p_{\mathsf{i}+k}$ are called *indexed propositions*. *Iteration bodies* are defined by the following grammar:

$$ib ::= \top \mid \neg ib \mid ib \wedge ib \mid p_{\mathsf{i}+k}$$

We can then define the grammar of *sequential propositional schemata*, or *SPS*, as follows:

$$sps ::= \top \mid \neg sps \mid sps \wedge sps \mid p_{\mathsf{n}+k} \mid p_k \mid \textstyle\bigwedge_{\mathsf{i}=0}^{\mathsf{n}-1} ib$$

An expression of the form $\bigwedge_{\mathsf{i}=0}^{\mathsf{n}-1} ib$ is called an *iteration*. We define $\bigvee_{\mathsf{i}=0}^{\mathsf{n}-1} ib$ as $\neg\bigwedge_{\mathsf{i}=0}^{\mathsf{n}-1} \neg ib$. We define $\vee$, $\Rightarrow$ and $\Leftrightarrow$ as usual (both for iteration bodies and for SPS). For instance, $p_0 \wedge \bigwedge_{\mathsf{i}=0}^{\mathsf{n}-1}(p_{\mathsf{i}} \Rightarrow p_{\mathsf{i}+1}) \wedge \neg p_{\mathsf{n}}$ or $\bigwedge_{\mathsf{i}=0}^{\mathsf{n}-1} p_{\mathsf{i}} \wedge \bigvee_{\mathsf{i}=0}^{\mathsf{n}-1} \neg p_{\mathsf{i}}$ are SPS. The essential point of schemata is that iterations are *symbolic expressions*: n is a formal parameter, not a meta variable denoting any number. The specificity of SPS is that they represent a structure which is sequentially repeated, n being considered as the length of the sequence.

An SPS is interpreted by first giving a value to n – which gives raise to a propositional formula ϕ, called an "instance" of the SPS – and then by giving a value to the propositional variables of ϕ. Note that a (non-trivial) SPS has an infinite set of instances. The substitution of $x \in \{\mathsf{n}, \mathsf{i}\}$ by $m \in \mathbb{N}$ is written $[m/x]$. The application of a substitution to indexed propositions is defined by: $p_{\mathsf{n}+k}[m/\mathsf{n}] \stackrel{\text{def}}{=} p_{m+k}$, $p_{\mathsf{i}+k}[m/\mathsf{i}] \stackrel{\text{def}}{=} p_{m+k}$ and in every other case the substitution has no effect. Applying a substitution to an iteration body is trivially done by propagating the substitution to the atoms (there is no problem of capture since no connective inside an iteration body can bind a variable). Then, let s be an SPS and $m \in \mathbb{N}$. The *instance* of s w.r.t. m is the propositional formula $\langle s \rangle_m$ inductively defined by:

$$\langle p_e \rangle_m \stackrel{\text{def}}{=} p_e[m/\mathsf{n}] \qquad \langle \neg s \rangle_m \stackrel{\text{def}}{=} \neg \langle s \rangle_m$$

$$\langle \textstyle\bigwedge_{\mathsf{i}=0}^{\mathsf{n}-1} s \rangle_m \stackrel{\text{def}}{=} \top \text{ if } m = 0 \qquad \langle s_1 \wedge s_2 \rangle_m \stackrel{\text{def}}{=} \langle s_1 \rangle_m \wedge \langle s_2 \rangle_m$$

$$\langle \textstyle\bigwedge_{\mathsf{i}=0}^{\mathsf{n}-1} s \rangle_m \stackrel{\text{def}}{=} \langle s[0/\mathsf{i}] \rangle_m \wedge \ldots \wedge \langle s[m-1/\mathsf{i}] \rangle_m \text{ if } m > 0$$

For example, the following are the instances of $p_0 \wedge \bigwedge_{\mathsf{i}=0}^{\mathsf{n}-1}(p_{\mathsf{i}} \Rightarrow p_{\mathsf{i}+1}) \wedge \neg p_{\mathsf{n}}$ for $m = 0, 1, 2$, respectively: $p_0 \wedge \neg p_0$, $p_0 \wedge (p_0 \Rightarrow p_1) \wedge \neg p_1$, $p_0 \wedge (p_0 \Rightarrow p_1) \wedge (p_1 \Rightarrow p_2) \wedge \neg p_2$. An instance is a usual propositional formula except that each variable is indexed with a natural number. So we just need a propositional interpretation to interpret this formula: let ϕ be a propositional formula whose variables are indexed by natural numbers, and σ a propositional interpretation. Then $\sigma \models \phi$ is defined as usual by induction on the structure of ϕ except that, for any indexed variable p_k, $\sigma \models p_k$ iff $(p, k) \in \sigma$. We thus define a *schema interpretation* as a pair consisting of a propositional interpretation and a natural number. Then a schema s is *true in a schema interpretation* $\mathfrak{I} = (\sigma, n)$ iff $\sigma \models \langle s \rangle_n$. We also write $\mathfrak{I} \models s$. Contrarily to general schemata, the satisfiability problem for SPS is decidable [4].

[1] As usual, k may be encoded in unary or in binary, as a sequence of digits. The choice between the two encodings has a significant influence on the complexity of the translations.

3. Translating interpretations

In the next sections we will see translations of LTL formulae into SPS and conversely. Some semantic translations underlie those syntactic ones. We make them explicit now in order to give preliminary insights.

3.1. From schemata to LTL

Consider a schema interpretation (σ, n). Given a schema interpretation (σ, n), its first component σ can already be considered as an LTL interpretation, but we still need to represent the second component n. This is done by using special LTL interpretations (which are also propositional interpretations) called "initial segments":

Definition 1. Let σ be a propositional interpretation over a set of variables $\mathcal{P}$. σ is an *initial segment* of length $k \in \mathbb{N}$ for some $p \in \mathcal{P}$ iff $(p, t) \in \sigma \Leftrightarrow t < k$.

The key feature of initial segments is that they can be put in correspondence with natural numbers. Namely, we can associate a canonical initial segment to every natural number and a natural number to every initial segment. This correspondence allows us to define the following transformation for schema interpretations:

Definition 2. Let $\mathcal{P}$ be a set of propositional variables and let "$\mathrm{t} < \mathsf{n}$" $\notin \mathcal{P}$ be a propositional variable. Let $\mathfrak{I} = (\sigma, n)$ be a schema interpretation over $\mathcal{P}$. Then $[\![\mathfrak{I}]\!]$ is the LTL interpretation over $\mathcal{P} \cup \{\mathrm{t} < \mathsf{n}\}$ which is an initial segment of length n for $\mathrm{t} < \mathsf{n}$ and which is defined as σ over $\mathcal{P}$. Conversely, $[\![.]\!]^{-1}$ is the function that maps every initial segment σ of length n for $\mathrm{t} < \mathsf{n}$ to the schema interpretation (τ, n) where τ is the restriction of σ to $\mathcal{P}$.

For instance, let $\mathfrak{I}$ be the schema interpretation $(\{p_0, q_0, p_1, p_2, q_3\}, 3)$. Then $[\![\mathfrak{I}]\!] = \{p, q, \mathrm{t} < \mathsf{n}\} \rightarrow \{p, \mathrm{t} < \mathsf{n}\} \rightarrow \{p, \mathrm{t} < \mathsf{n}\} \rightarrow \{q\} \rightarrow \{\} \rightarrow \{\} \rightarrow \ldots$ Conversely, let σ be the LTL interpretation $\{q, \mathrm{t} < \mathsf{n}\} \rightarrow \{q, \mathrm{t} < \mathsf{n}\} \rightarrow \{p, \mathrm{t} < \mathsf{n}\} \rightarrow$

$\{p, q, \mathrm{t} < \mathsf{n}\} \rightarrow \{p\} \rightarrow \{p\} \rightarrow \ldots$, then $\llbracket\sigma\rrbracket^{-1} = (\{q_0, q_1, p_2, p_3, q_3, p_4, p_5, \ldots\}, 4)$.

The map $\llbracket.\rrbracket$ is a bijection between schema interpretations over $\mathcal{P}$ and initial segments over $\mathcal{P} \cup \{\mathrm{t} < \mathsf{n}\}$. Indeed, $\llbracket.\rrbracket^{-1}$ is its inverse. Initial segments thus allow us to simulate finite models in LTL. The set of initial segments can be specified in LTL as follows:

Proposition 3. *Let* $\phi_<^{\mathrm{t}<\mathsf{n}} \stackrel{def}{=} (\mathrm{t} < \mathsf{n})\mathrm{UG}(\neg\mathrm{t} < \mathsf{n})$. *Then an LTL interpretation is a model of* $\phi_<^{\mathrm{t}<\mathsf{n}}$ *iff it is an initial segment for* $\mathrm{t} < \mathsf{n}$.

We can also specify a proposition eq^{n} true only at time n. This is axiomatized by: $\mathrm{Ax}_{\mathrm{t}=\mathsf{n}} \stackrel{\text{def}}{=} \mathrm{G}(\mathrm{t} < \mathsf{n} \wedge \neg\mathrm{X}(\mathrm{t} < \mathsf{n}) \Leftrightarrow \mathrm{X}(\mathrm{eq}^{\mathrm{n}})) \wedge (\neg\mathrm{t} < \mathsf{n} \Leftrightarrow \mathrm{eq}^{\mathrm{n}})$. To improve readability, eq^{n} will be written $\mathrm{t} = \mathsf{n}$. Let σ be an initial segment for $\mathrm{t} < \mathsf{n}$ of length n s.t. $\sigma, 0 \models \mathrm{Ax}_{\mathrm{t}=\mathsf{n}}$. Then it is easily shown that $\sigma, t \models \mathrm{t} = \mathsf{n}$ iff $t = n$.

3.2. From LTL to schemata

The inverse translation is harder: embedding LTL into schemata means that we must represent the infinite interpretations of LTL using only schema interpretations, which are finite. Of course this is impossible in general. However, as we are concerned with satisfiability, we can restrict ourselves to UP interpretations. Since those can be finitely represented, we will be able to embed them into schema interpretations. This is achieved via "2-initial segments":

Definition 4. A schema interpretation $\mathfrak{I} = (\sigma, n)$ is a 2-*initial segment* for a propositional variable p iff there is $k \leq n$ s.t., for every $l \in \{0, \ldots, n\}$, we have $(p, l) \in \sigma \Leftrightarrow l < k$. We call k the *short length* of $\mathfrak{I}$ and $n + 1$ is its *long length*.

For instance, the schema interpretation $(\{p_0, p_1, p_2\}, 5)$ is a 2-initial segment w.r.t. p. Its short length is 3, its long length is 6. We call this a 2-initial segment because two initial segments are characterized: $\{0, \ldots, k-1\}$ and $\{0, \ldots, n\}$. But notice that p is not specified above n. This is not a problem since we will not need such values in the translations. The notion of 2-initial segment is useful because, much in the same way in which initial segments correspond to natural numbers, 2-initial segments correspond to *pairs* of different natural numbers. We can now define the following transformation for UP interpretations:

Definition 5. Let σ be a UP interpretation of prefix index k and of period l over a set $\mathcal{P}$, and let "pfx" $\notin \mathcal{P}$ be a propositional variable. Then $\lceil\sigma\rceil$ is the schema interpretation $(\tau, k + l - 1)$ where τ is defined as an initial segment of length k for pfx and preserving the value of σ on $\mathcal{P}$.

Remark 6. The map $\lceil.\rceil$ embeds the prefix index and the period inside schema interpretations, but it is impossible to specify the fact that an interpretation is a UP interpretation: this would require to express that the interpretation loops *indefinitely*. Such an "infinite" behaviour cannot be specified with schemata. This will not be a problem in the following because, for a given LTL formula, one only needs to specify this behaviour in the range $\{0, \ldots, k + l - 1\}$.

For similar reasons, $\lceil.\rceil$ is *not* a bijection in general, unlike $\llbracket.\rrbracket$. It is actually a bijection between UP interpretations and 2-initial segments if we restrict the latter to the values assigned to variables whose index is between 0 and $k+l-1$. This will indeed be the case in our reduction since, as just explained, we will not need the values for other indices. Then $\lceil.\rceil^{-1}$ is defined as follows:

Definition 7. Let (σ, n) be a 2-initial segment for pfx. Then $\lceil\sigma, n\rceil^{-1}$ is defined as the unique UP interpretation such that: 1. its prefix is the set of instants s.t. pfx holds in $\mathfrak{I}$; 2. its period l is $n - k + 1$, where k is the prefix index; 3. for all $p \neq \mathrm{pfx}$ and all $t \leq n$, $(p, t) \in \lceil\mathfrak{I}, n\rceil^{-1}$ iff $(p, t) \in \mathfrak{I}$.

Proposition 8. *Let* $\mathrm{s}_\leq^{\mathrm{pfx}}$ *be the SPS* $\neg\mathrm{pfx}_{\mathsf{n}} \wedge \bigwedge_{\mathsf{i}=0}^{\mathsf{n}-1}(\mathrm{pfx}_{\mathsf{i}+1} \Rightarrow \mathrm{pfx}_{\mathsf{i}})$. *A schema interpretation is a model of* $\mathrm{s}_\leq^{\mathrm{pfx}}$ *iff it is a* 2*-initial segment for* pfx.

This proposition shows that 2-initial segments can be specified using schemata. The beginning of the loop can be referred to by using a propositional variable $\mathrm{eq}_{\mathsf{i}}^k$, intended to be true only when i is equal to the prefix index k of the interpretation. This can be axiomatized as follows: $\mathrm{Ax}_{\mathsf{i}=k} \stackrel{\text{def}}{=} (\neg\mathrm{pfx}_0 \Leftrightarrow \mathrm{eq}_0^k) \wedge \bigwedge_{\mathsf{i}=0}^{\mathsf{n}-1}(\mathrm{pfx}_{\mathsf{i}} \wedge \neg\mathrm{pfx}_{\mathsf{i}+1} \Leftrightarrow \mathrm{eq}_{\mathsf{i}+1}^k)$. To improve readability, $\mathrm{eq}_{\mathsf{i}}^k$ will be written "$\mathsf{i} = k$".

4. Embedding SPS in LTL

Now, given an SPS s, we build an LTL formula $\lfloor s\rfloor$ which is satisfiable iff s is satisfiable. The main desideratum of $\lfloor.\rfloor$ is that for every model $\mathfrak{M}$ of an SPS s, the interpretation $\llbracket\mathfrak{M}\rrbracket$ is a model of $\lfloor s\rfloor$. By Proposition 3, every interpretation s.t. $\phi_<^{\mathrm{t}<\mathsf{n}}$ holds is an initial segment of length n for a propositional variable "$\mathrm{t} < \mathsf{n}$". Furthermore, $\mathrm{Ax}_{\mathrm{t}=\mathsf{n}}$ enables to use the variable "$\mathrm{t} = \mathsf{n}$". Our translation thus includes those formulae.

Definition 9. Let s be an SPS. Then $\lfloor s\rfloor$ is an LTL formula defined as $\lfloor s\rfloor \stackrel{\text{def}}{=} \lfloor s\rfloor_{\mathrm{prop}} \wedge \phi_<^{\mathrm{t}<\mathsf{n}} \wedge \mathrm{Ax}_{\mathrm{t}=\mathsf{n}}$ where $\lfloor s\rfloor_{\mathrm{prop}}$ is inductively defined as follows:

$$\begin{aligned}
\lfloor\top\rfloor_{\mathrm{prop}} &\stackrel{\text{def}}{=} \top \\
\lfloor p_k\rfloor_{\mathrm{prop}} &\stackrel{\text{def}}{=} \mathrm{X}^k p \\
\lfloor p_{\mathsf{n}+k}\rfloor_{\mathrm{prop}} &\stackrel{\text{def}}{=} \mathrm{G}(\mathrm{t} = \mathsf{n} \Rightarrow \mathrm{X}^k p) \\
\lfloor p_{\mathsf{i}+k}\rfloor_{\mathrm{prop}} &\stackrel{\text{def}}{=} \mathrm{X}^k p \\
\lfloor\neg s\rfloor_{\mathrm{prop}} &\stackrel{\text{def}}{=} \neg\lfloor s\rfloor_{\mathrm{prop}}
\end{aligned}$$

$$\lfloor s_1 \wedge s_2 \rfloor_{\text{prop}} \stackrel{\text{def}}{=} \lfloor s_1 \rfloor_{\text{prop}} \wedge \lfloor s_2 \rfloor_{\text{prop}}$$
$$\lfloor \bigwedge_{\mathsf{i}=0}^{\mathsf{n}-1} s \rfloor_{\text{prop}} \stackrel{\text{def}}{=} \mathrm{G}(\mathrm{t} < \mathsf{n} \Rightarrow \lfloor s \rfloor_{\text{prop}})$$

where $k \in \mathbb{N}$, $\mathsf{i} \neq \mathsf{n}$, and $\mathrm{X}^k\phi$ is $\mathrm{X}\ldots\mathrm{X}\phi$ with k X's.

For instance, $\lfloor p_0 \wedge \bigwedge_{\mathsf{i}=0}^{\mathsf{n}-1}(p_\mathsf{i} \Rightarrow p_{\mathsf{i}+1}) \wedge \neg p_\mathsf{n} \rfloor = p \wedge \mathrm{G}(\mathrm{t} < \mathsf{n} \Rightarrow p \Rightarrow \mathrm{X}p) \wedge \neg\mathrm{G}(\mathrm{t} = \mathsf{n} \Rightarrow p) \wedge \phi_<^{\mathrm{t}<\mathsf{n}} \wedge \mathrm{Ax}_{\mathrm{t}=\mathsf{n}}$. Note that, although other encodings are of course possible, the use of the connective G (or, more generally, U) cannot, of course, be avoided. We have then:

Theorem 10. *Let s be a SPS. Then $[\![.]\!]$ is a bijection between the models of s and the models of $\lfloor s \rfloor$. The inverse bijection is $[\![.]\!]^{-1}$.*

An obvious consequence is that s is satisfiable iff $\lfloor s \rfloor$ is satisfiable. However our result is more interesting since it provides more insights about the translation and makes explicit the inverse transformation for interpretations, which is useful for model building. Consequently we can use any LTL satisfiability solver to solve the satisfiability problem for SPS: we simply translate the input schema to LTL with $\lfloor . \rfloor$ and then launch the LTL solver on the output formula. Thus the satisfiability problem for SPS can be reduced to the satisfiability problem for LTL. Notice furthermore that if the solver finds a model, then we can translate it back to a schema model using the inverse translation $[\![.]\!]^{-1}$.

We can easily study the complexity of this transformation. Let $\#s$ denote the size of a schema s, in number of symbols, and let $\#_{\text{int}}s$ denote the size of the biggest number occurring in s, expressed w.r.t. the size of s. This is to take into account the fact that numbers can be encoded either in unary or in binary: if they are encoded in binary then $\#_{\text{int}}s = O(2^{\#s})$, but if they are encoded in unary then $\#_{\text{int}}s = O(\#s)$. It may also happen that we consider only schemata whose biggest number is bounded by some constant; in such a case, we have $\#_{\text{int}}s = O(1)$. This case is worth considering since we may increase the size of a schema without increasing the numbers that occur in it. Then, for every SPS s, we have $\#\lfloor s \rfloor = O(\#s.\#_{\text{int}}s)$. Consequently, $\lfloor . \rfloor$ is linear if numbers are bounded by constants, quadratic if numbers are encoded in unary, exponential if they are encoded in binary. Since the satisfiability of LTL is in PSPACE [26]:

Theorem 11. *The satisfiability of SPS is in PSPACE (resp. EXPSPACE) if numbers are encoded in unary or bounded by constants (resp. coded in binary).*

5. Embedding LTL in SPS

We now tackle the reverse embedding. We need the UP model property to obtain a successful translation, written $\lceil . \rceil$, of LTL formulae into SPS. The aim of $\lceil . \rceil$ is that for every model σ of an LTL formula ϕ, the interpretation $\llbracket \sigma \rrbracket$ (Definition 5) is a model of $\lceil \phi \rceil$. This transformation uses a structure-preserving approach: for each subformula ϕ (different from an indexed proposition) of the original formula, we introduce a fresh propositional variable written $|\phi|$. For an indexed proposition p, $|p| \stackrel{\text{def}}{=} p$. Each indexed propositional variable $|\phi|_\mathsf{i}$, $0 \leq \mathsf{i} \leq \mathsf{n}$, is then intended to be true iff the subformula ϕ is true at time i. Formally, we extend $\llbracket . \rrbracket$ as follows:

Definition 12. Let σ be a UP interpretation and ϕ an LTL formula. Then: for every propositional variable of the form $|\psi|$ for some subformula ψ of ϕ, $(|\psi|, t) \in \llbracket \sigma \rrbracket$ iff $\sigma, t \models \psi$; for every other variable, $\llbracket \sigma \rrbracket$ is defined as described early on.

Furthermore, for each subformula of the form $\phi_1 \mathrm{U} \phi_2$, we add another propositional variable called $|\phi_1 \mathrm{U}' \phi_2|$ (called this way because its behaviour is very close to the one of U) interpreted as **T** at $t \in \mathbb{N}$ iff there is $t' \in \mathbb{N}$ s.t. $t \leq t' \leq k + l - 1$ where ϕ_1 holds between t and $t' - 1$ and ϕ_2 holds at t', i.e. the semantics are the same as for U except that the instant when ϕ_2 occurs must happen before the end of the loop (as explained thereafter, this variable is used to ensure that the eventuality indeed happens).

The inverse operation is defined as in Definition 7 except that the value of any variable $|\psi|$ is "forgotten". The translation is done by adding axioms to compute the values of the newly introduced propositional variables (relating these values to the ones of the propositional variables originally occurring in the formula). As we shall see, the specification of those new variables is straightforward when the head symbol of the subformula is a boolean connective: the value of the considered variable can be directly related to the values of the variables corresponding to the operands, see definition of $\mathrm{Ax}_{\neg\phi}$ and $\mathrm{Ax}_{\phi_1 \wedge \phi_2}$ in Definition 13 below.

When the head symbol of the subformula is a temporal connective, we have to distinguish whether the index denotes a time lower or equal to n (since the interpretation is UP, we only have to consider the time interval $\{0, \ldots, \mathsf{n}\}$). In both cases, the value of the considered propositional variable $|\phi|$ at time i is related to the one of the variables *at the next instant*. If $\mathsf{i} < \mathsf{n}$ then this next instant is easy to compute: it is simply $\mathsf{i} + 1$. But if $\mathsf{i} = \mathsf{n}$, since the value of the variables $|\phi|$ are specified only on the interval $\{0, \ldots, \mathsf{n}\}$ we cannot refer to the time $\mathsf{n} + 1$ and we have to take advantage of the fact that the interpretation is periodic: since n necessarily corresponds to the end of the periodic part, the next instant must be the beginning of the loop. This is easily handled in the X case: if we have $\mathrm{X}\phi$ at time n then we must have ϕ at time k where k is the beginning of the loop.

In the U case, if we have $\phi_1 \mathrm{U} \phi_2$ at time n then we have to deal with the fact that ϕ_2 might hold after n, between time

k and $\mathsf{n}-1$ (by taking the loop into account). In this case we have to check that ϕ_2 holds between k and $\mathsf{n}-1$, and that ϕ_1 holds in between. This check is triggered by the use of the new connective U', whose specification is thus added to the definition. Intuitively, $\phi_1 \mathrm{U}' \phi_2$ may be seen as a connective interpreted as $\phi_1 \mathrm{U} \phi_2$, except that the formula ϕ_2 must hold *at the latest* at time n (notice that using U instead of U' would yield an ill-founded definition: the eventuality could be always delayed and never fulfilled).

Definition 13. Let ϕ be an LTL formula. Then $\lceil \phi \rceil$ is $\lceil \phi \rceil \stackrel{\text{def}}{=} |\phi|_0 \wedge \Phi^\phi \wedge \mathrm{s}^{\mathrm{pfx}}_{\leq} \wedge \mathrm{Ax}_{\mathsf{i}=k}$ where Φ^ϕ stands for $\bigwedge\{\mathrm{Ax}_\psi \mid \psi \text{ is a subformula of } \phi\}$ and Ax_ψ is defined by:

$$\mathrm{Ax}_\top \stackrel{\text{def}}{=} \textstyle\bigwedge_{\mathsf{i}=0}^{\mathsf{n}} |\top|_{\mathsf{i}} \quad \mathrm{Ax}_{\neg\phi} \stackrel{\text{def}}{=} \bigwedge_{\mathsf{i}=0}^{\mathsf{n}} (|\neg\phi|_{\mathsf{i}} \Leftrightarrow \neg |\phi|_{\mathsf{i}})$$
$$\mathrm{Ax}_{\phi_1 \wedge \phi_2} \stackrel{\text{def}}{=} \bigwedge_{\mathsf{i}=0}^{\mathsf{n}} (|\phi_1 \wedge \phi_2|_{\mathsf{i}} \Leftrightarrow |\phi_1|_{\mathsf{i}} \wedge |\phi_2|_{\mathsf{i}})$$
$$\mathrm{Ax}_{\mathrm{X}\phi} \stackrel{\text{def}}{=} \bigwedge_{\mathsf{i}=0}^{\mathsf{n}-1} (|\mathrm{X}\phi|_{\mathsf{i}} \Leftrightarrow |\phi|_{\mathsf{i}+1})$$
$$\wedge (|\mathrm{X}\phi|_{\mathsf{n}} \Leftrightarrow \bigwedge_{\mathsf{i}=0}^{\mathsf{n}} (\mathsf{i} = k \Rightarrow |\phi|_{\mathsf{i}}))$$

and $\mathrm{Ax}_{\phi_1 \mathrm{U} \phi_2}$ is the conjunction of the following formulae:

$$\textstyle\bigwedge_{\mathsf{i}=0}^{\mathsf{n}-1} (|\phi_1 \mathrm{U} \phi_2|_{\mathsf{i}} \Leftrightarrow |\phi_2|_{\mathsf{i}} \vee (|\phi_1|_{\mathsf{i}} \wedge |\phi_1 \mathrm{U} \phi_2|_{\mathsf{i}+1}))$$
$$|\phi_1 \mathrm{U} \phi_2|_{\mathsf{n}} \Leftrightarrow (|\phi_2|_{\mathsf{n}} \vee |\phi_1|_{\mathsf{n}} \wedge \textstyle\bigwedge_{\mathsf{i}=0}^{\mathsf{n}} (\mathsf{i} = k \Rightarrow |\phi_1 \mathrm{U}' \phi_2|_{\mathsf{i}}))$$
$$\textstyle\bigwedge_{\mathsf{i}=0}^{\mathsf{n}-1} (|\phi_1 \mathrm{U}' \phi_2|_{\mathsf{i}} \Leftrightarrow |\phi_2|_{\mathsf{i}} \vee (|\phi_1|_{\mathsf{i}} \wedge |\phi_1 \mathrm{U}' \phi_2|_{\mathsf{i}+1}))$$
$$|\phi_1 \mathrm{U}' \phi_2|_{\mathsf{n}} \Leftrightarrow |\phi_2|_{\mathsf{n}}$$

where $\bigwedge_{\mathsf{i}=0}^{\mathsf{n}} s$ is a shortcut for $\bigwedge_{\mathsf{i}=0}^{\mathsf{n}-1} s \wedge s[\mathsf{n}/\mathsf{i}]$ (we need to define this as an abbreviation so that the schema be indeed sequential).

Theorem 14. *Let ϕ be an LTL formula. Then $[\![.]\!]$ is a bijection between UP models of ϕ and models of $\lceil \phi \rceil$ (if the latter are restricted to the values of variables occurring in the corresponding instance). $[\![.]\!]^{-1}$ is the inverse bijection.*

It is trivial that $\#\lceil \phi \rceil$ is linear w.r.t. $\#\phi$. Thus:

Theorem 15. *The satisfiability problem for SPS is PSPACE-complete if numbers are encoded in unary or bounded by a constant.*

For practical efficiency, we can improve over Definition 13. We can translate the purely propositional connectives directly, i.e. without axiomatising them: any occurrence of an atom $|\top|_e$ (resp. $|\neg\phi|_e$, resp. $|\phi_1 \wedge \phi_2|_e$) is directly replaced by $\top$ (resp. $\neg|\phi|_e$, resp. $|\phi_1|_e \wedge |\phi_2|_e$) repeatedly until there is no more such occurrence. The same applies to $\vee$, $\Rightarrow$ and $\Leftrightarrow$. Those are defined as abbreviations in the present paper in order to simplify definitions, but it is of course more efficient in practice to translate them directly when available as primitive connectives (obviously, this is also true for Definition 9).

Another optimization can be devised by observing that all schemata decision procedures [1, 2] reason by induction on n, i.e. they refute a schema for any value of n by reduction to the case $\mathsf{n}-1$. In our reduction, n corresponds to the last instant of the UP interpretation. Consequently, a schema procedure applied to a translated LTL formula starts by considering the *last* instant of the interpretation and then going backward. This is counter natural since we try to refute a formula *at time* 0. To tackle this problem we just need to change the translation by "inverting the time": i.e. the index 0 will be interpreted as the last instant of the period and the index n as its first instant. Concretely, in Definition 13, we just rewrite every index $\mathsf{i}-1$ into i, every index i into $\mathsf{i}+1$, every index 0 into n, and every index n into 0. Experiments with this translation indeed confirm that conjectures are refuted faster using this new translation.

Remark 16. The translation given here might remind the reader of bounded model checking (BMC) [9]. A very important difference however is that our reduction is *complete*, which is of course not the case of BMC. Indeed, the whole point of schemata is to reason about an infinite family of propositional formulae *without having to instantiate the parameter*. Our translation could of course be used for BMC, simply by instantiating the parameter with successive natural numbers. However the converse does not hold: not every translation found in BMC could fit instead of Definition 13, since the result must respect the syntactical criteria ensuring decidability of the satisfiability problem. For instance, renaming sub-formulae by propositional variables is just an optimization in the case of BMC whereas in our case, it is *needed* since, otherwise, the resulting schema would not be sequential (and not even regular). Completeness is an important problem in BMC which is usually tackled with notions like completeness thresholds and recurrence diameter [9] or induction [25]. Thorough analysis of how schemata procedures handle the above translation could give new ideas in order to get completeness for BMC.

6. Implementation

The implementations of both translations are available at `http://membres-liglab.imag.fr/aravantinos/Site/Software.html`. Some preliminary experiments have been achieved on a few benchmarks: standard schemata examples provided with RegSTAB [3] have been translated to LTL (note that the examples have been slightly modified in order to fit the constraints of SPS) and standard LTL pattern formulae [23] have been translated to SPS. The performance of RegSTAB and `pltl` (`http://users.cecs.anu.edu.au/~rpg/software.html`) have been compared on both benchmarks. *In both cases,* `pltl` clearly outperformed RegSTAB. We see two reasons to this:

- RegSTAB deals with regular schemata, which are more general than SPS. In particular, the decision procedure

for such schemata requires the detection and elimination of pure literals (an adaptation of the "Affirmative-negative rule" of [12]), which is well-known to be a huge time-consuming task (and this is even more the case for schemata since we have to deal with a *symbolic* notion of pure literal). This auxiliary procedure is needed for termination, and is mainly a consequence of the "non-local" aspect of schemata.

- With LTL procedures, given a formula ϕ, one knows in advance all the formulae that will occur in the deduction process: all of them belong to the closure of ϕ (merely the set of all subformulae of ϕ, closed by negation and unfolding of temporal formulae); this permits the use of efficient data structures to represent sets of formulae, e.g. `pltl` uses bitsets. This is not the case of SPS (and even more regular schemata), e.g. refuting a schema containing $\bigwedge_{\mathsf{i}=0}^{\mathsf{n}} p_{\mathsf{i}}$ potentially leads to the introduction of p_{n}, $p_{\mathsf{n}-1}$, $p_{\mathsf{n}-2}$, etc. By termination for regular schemata [1], this enumeration is finite but one does not know in advance how far it has to go. Hence the data structures used in RegSTAB are much heavier: e.g. we use balanced trees for sets of formulae. Thus, for big examples, the memory is easily saturated and RegSTAB spends much of its time in its handling which was absolutely not the case of `pltl`.

The most important reason seems to be the second one. It can actually be tackled in order to improve RegSTAB performance: we can syntactically extract from the input schema a bound for the above enumeration p_{n}, $p_{\mathsf{n}-1}$, $p_{\mathsf{n}-2}$, ... by analysis of the termination proof for regular schemata. Implementing this technique is ongoing work.

Yet, there are examples where RegSTAB did better than `pltl`. Consider $(p_1 \Rightarrow q_{\mathsf{n}+1}) \wedge p_1 \wedge \neg q_{\mathsf{n}+1} \wedge \phi$ where ϕ is any formula involving some iterations. This schema is immediately refuted by RegSTAB, but the bigger ϕ is, the longer it takes for `pltl` to refute the corresponding LTL formula. Of course, this example was devised to emphasize one of the strengths of RegSTAB: contrarily to LTL procedures in general, and to `pltl` in particular, reasoning about schemata is *global*, i.e. RegSTAB may reason simultaneously on propositions containing various *symbolic* indices. In contrast, `pltl` will analyse the formula ϕ and the contradiction will appear only at the end of the construction (i.e. by "discovering" eventually that $t = \mathsf{n}$ cannot hold at any state, since it would allow to derive a contradiction).

7. Conclusion and future work

LTL formulae and SPS have been shown to be reducible to each other in polynomial time (exponential time when numbers are encoded in binary). The reduction of SPS to LTL is unsurprising. The converse reduction makes use of the well-known fact that the infinite semantics of LTL can be finitely represented. This entails that the satisfiability of SPS is PSPACE-complete.

Pros and cons of each logic. Since LTL and SPS are equivalent w.r.t. satisfiability, one may wonder which to favour. There are two major differences between LTL and schemata: first, LTL default interpretations are *infinite* whereas those of schemata are *finite*; second, LTL refers to states in an *anonymous* way, whereas schemata *name* them. These differences provide us with clear criteria for choosing one logic or the other in different situations: to specify an infinite behaviour, one would naturally use LTL, whereas classes of structurally similar finite behaviours are more naturally specified with schemata. Unsurprisingly, the specification of temporal behaviours falls of course in the first category. But, e.g., the specification of a circuit independently of the number of bits of its input falls in the second category. Consider for instance the specification of a ripple-carry adder: $\bigwedge_{\mathsf{i}=0}^{\mathsf{n}}((s_{\mathsf{i}} \Leftrightarrow (x_{\mathsf{i}} \oplus y_{\mathsf{i}}) \oplus c_{\mathsf{i}}) \wedge (c_{\mathsf{i}+1} \Leftrightarrow (x_{\mathsf{i}} \wedge y_{\mathsf{i}}) \vee (y_{\mathsf{i}} \wedge c_{\mathsf{i}}) \vee (x_{\mathsf{i}} \wedge c_{\mathsf{i}}))) \wedge \neg c_0$, where $x_0, \ldots, x_{\mathsf{n}}$ and $y_0, \ldots, y_{\mathsf{n}}$ are the input bit vectors of size n; $s_0, \ldots, s_{\mathsf{n}}$ is the output bit vector and $c_0, \ldots, c_{\mathsf{n}}$ is the carry vector. Here the indices indeed correspond to the time in a *concrete* sequential circuit. But from a specification point of view, those indices are just an abstract way to represent a generic scheme of circuits. Consequently, the schema syntax seems better suited to this case (and notice that it is very intuitive).

Similarly, the choice between a named or an anonymous representation of states depends on the situation. The X connective is well suited to express properties in a *local* way, since there is no need to explicitly use an index to refer to the current or the next state. The U connective is also far more intuitive than its translation to SPS to refer to *some instant* satisfying some property in the future. On the other hand, in order to refer to an *identified* instant of the future, one needs to refer to it by giving it a name, which is easily done with the schema syntax. Consider e.g. the example $p_0 \wedge \bigwedge_{\mathsf{i}=0}^{\mathsf{n}-1}(p_{\mathsf{i}} \Rightarrow p_{\mathsf{i}+1}) \wedge \neg p_{\mathsf{n}}$ translated as $p \wedge \mathrm{G}(\mathrm{t} < \mathsf{n} \Rightarrow p \Rightarrow \mathrm{X}p) \wedge \mathrm{G}(\mathrm{t} = \mathsf{n} \Rightarrow \neg p)$ (plus the necessary axioms $\phi_{<}^{\mathrm{t}<\mathsf{n}} \wedge \mathrm{Ax}_{\mathrm{t}=\mathsf{n}}$) in LTL. One can even specify behaviours *after* that time (but this goes beyond sequential schemata [4]), e.g. one can write $p_0 \wedge \bigwedge_{\mathsf{i}=0}^{\mathsf{n}}(p_{\mathsf{i}} \Rightarrow p_{\mathsf{i}+1}) \wedge \bigwedge_{\mathsf{i}=\mathsf{n}}^{2\mathsf{n}}(\neg p_{\mathsf{i}+1} \Rightarrow \neg p_{\mathsf{i}}) \wedge \neg p_{2\mathsf{n}}$. It seems improbable that such a property would be useful in a temporal context, but this could be used to specify planning problems with some predefined strategy e.g. if one wants to allow some set of actions in a first phase of a planning problem and then another set in some other phase of this problem.

Future work. Using the above translations to help export procedures from one logic to another is an obvious follow-up of this work (in particular, DPLL inspired procedures for schemata could help defining such a procedure for LTL). Similarly, as explained in Remark 16, investigating how

model checking is done by translation to schemata could give ideas to define new completeness criteria for bounded model checking. The extension of the presented results to other classes of schemata could also be considered, e.g. schemata with nested iterations (proved decidable in [2, 4]). Translation algorithms from nested schemata into sequential ones exist [4], however they are of double exponential complexity. Thus we conjecture that no polynomial-time transformation from nested schemata to LTL exists. The extension of this study to other – more expressive – temporal logics could also be of interest. Notably, LTL with past operators [21] seems to be easily handled with (non sequential) schemata simply by allowing negative numbers in indices. Since implementations for this logic do not have the same support as standard LTL and are generally not as efficient, such a reduction could help in improving those points. One could go even further by making connections between schemata and monadic second order logic (MSO). This would be interesting both in theory and practice, since few implementations are available for MSO (only MONA [19] seems to be actively maintained).

References

[1] V. Aravantinos, R. Caferra, and N. Peltier. A Schemata Calculus for Propositional Logic. In *TABLEAUX*, volume 5607, pages 32–46. Springer, 2009.

[2] V. Aravantinos, R. Caferra, and N. Peltier. A Decidable Class of Nested Iterated Schemata. In Giesl and Hähnle [18], pages 293–308.

[3] V. Aravantinos, R. Caferra, and N. Peltier. RegSTAB: A SAT-Solver for Propositional Iterated Schemata. In Giesl and Hähnle [18], pages 309–315.

[4] V. Aravantinos, R. Caferra, and N. Peltier. Decidability and Undecidability Results for Propositional Schemata. *Journal of Artificial Intelligence Research*, 40:599–656, 2011.

[5] F. Bacchus and F. Kabanza. Using Temporal Logic to Control Search in a Forward Chaining Planner. In *3rd European Workshop on Planning*, pages 141–153. Press, 1995.

[6] J. A. Baier and S. A. Mcilraith. Planning with first-order temporally extended goals using heuristic search. In *National Conference on Artificial Intelligence*, pages 788–795. AAAI Press, 2006.

[7] P. Balsiger, A. Heuerding, and S. Schwendimann. Logics Workbench 1.0. In H. C. M. de Swart, editor, *TABLEAUX*, volume 1397, pages 35–37. Springer, 1998.

[8] A. Bauer and P. Haslum. LTL Goal Specifications Revisited. In *ECAI*, pages 881–886, Amsterdam, Aug 2010. IOS Press.

[9] A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu. Bounded model checking. *Advances in Computers*, 58:118–149, 2003.

[10] H. Calbrix, M. Nivat, and A. Podelski. Ultimately Periodic Words of Rational ω-Languages. In *MFPS 1994*, pages 554–566, London, UK, 1994. Springer-Verlag.

[11] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In E. Brinksma and K. G. Larsen, editors, *CAV*, volume 2404, pages 359–364. Springer, 2002.

[12] M. Davis and H. Putnam. A Computing Procedure for Quantification Theory. *J. ACM*, 7:201–215, July 1960.

[13] M. De Wulf, L. Doyen, N. Maquet, and J. F. Raskin. Antichains: alternative algorithms for LTL satisfiability and model-checking. In *TACAS'08/ETAPS'08*, pages 63–77, Berlin, Heidelberg, 2008. Springer-Verlag.

[14] C. Eisner, D. Fisman, J. Havlicek, Y. Lustig, A. McIsaac, and D. V. Campenhout. Reasoning with Temporal Logic on Truncated Paths. In W. A. H. Jr. and F. Somenzi, editors, *CAV*, volume 2725, pages 27–39. Springer, 2003.

[15] M. Fisher, C. Dixon, and M. Peim. Clausal temporal resolution. *ACM Trans. Comput. Logic*, 2:12–56, January 2001.

[16] D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. In *POPL*, pages 163–173, New York, NY, USA, 1980. ACM.

[17] O. Gasquet, A. Herzig, D. Longin, and M. Sahade. LoTREC: Logical Tableaux Research Engineering Companion. In B. Beckert, editor, *TABLEAUX*, volume 3702, pages 318–322. Springer Berlin / Heidelberg, 2005.

[18] J. Giesl and R. Hähnle, editors. *IJCAR*, volume 6173. Springer, 2010.

[19] J. Henriksen, J. Jensen, M. Jørgensen, N. Klarlund, B. Paige, T. Rauhe, and A. Sandholm. Mona: Monadic Second-order logic in practice. In *TACAS '95, LNCS 1019*, 1995.

[20] U. Hustadt and B. Konev. TRP++2.0: A Temporal Resolution Prover. In F. Baader, editor, *CADE*, volume 2741, pages 274–278. Springer, 2003.

[21] O. Lichtenstein, A. Pnueli, and L. D. Zuck. The Glory of the Past. In *CLP*, pages 196–218, London, UK, 1985. Springer-Verlag.

[22] A. Pnueli. The temporal logic of programs. In *Proceedings of FOCS 1977*, pages 46–57, Washington, DC, USA, 1977. IEEE Computer Society.

[23] K. Y. Rozier and M. Y. Vardi. LTL satisfiability checking. In *Proceedings of the 14th international SPIN conference on Model checking software*, pages 149–167, Berlin, Heidelberg, 2007. Springer-Verlag.

[24] S. Schwendimann. A New One-Pass Tableau Calculus for PLTL. In H. de Swart, editor, *TABLEAUX*, volume 1397, pages 277–291. Springer Berlin / Heidelberg, 1998.

[25] M. Sheeran, S. Singh, and G. Stålmarck. Checking safety properties using induction and a sat-solver. FMCAD '00, pages 108–125, London, UK, 2000. Springer-Verlag.

[26] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, 1985.

[27] W. Thomas. Star-free regular sets of ω-sequences. *Information and Control*, 42(2):148 – 156, 1979.

[28] P. Wolper. The tableau method for temporal logic: an overview. *Logique et Analyse*, 28:119–136, 1985.

[29] P. Wolper, M. Y. Vardi, and A. P. Sistla. Reasoning about infinite computation paths. *Foundations of Computer Science, Annual IEEE Symposium on*, 0:185–194, 1983.