A (non-elementary) modular decision procedure for LTrL

Paul Gastin[1], Raphaël Meyer[2] and Antoine Petit[2]

[1] LIAFA, Université Paris 7, 2, place Jussieu, F-75251 Paris Cedex 05
{Paul.Gastin}@liafa.jussieu.fr
[2] LSV, URA 2236 CNRS, ENS de Cachan, 61, av. du Prés. Wilson, F-94235 Cachan Cedex
{rmeyer,petit}@lsv.ens-cachan.fr

Abstract. Thiagarajan and Walukiewicz [18] have defined a temporal logic LTrL on Mazurkiewicz traces, patterned on the famous propositional temporal logic of linear time LTL defined by Pnueli. They have shown that this logic is equal in expressive power to the first order theory of finite and infinite traces.
The hopes to get an "easy" decision procedure for LTrL, as it is the case for LTL, vanished very recently due to a result of Walukiewicz [19] who showed that the decision procedure for LTrL is non-elementary.
However, tools like Mona [8] or Mosel [7] show that it is possible to handle non-elementary logics on significant examples.
Therefore, it appears worthwhile to have a direct decision procedure for LTrL; in this paper we propose such a decision procedure, in a modular way. Since the logic LTrL is not pure future, our algorithm constructs by induction a finite family of Büchi automata for each LTrL-formula. As expected by the results of [19], the main difficulty comes from the "Until" operator.

Topics: logic in computer science, automata and formal languages, theory of parallel and distributed computation, model-checking

1 Introduction

A run of a distributed system can be viewed, in many settings, as a partial order between the events of the system. Two events are ordered if and only if their executions depend causally one of the other. The partial orders that arise in this fashion, are frequently Mazurkiewicz traces [9,4]. A major interest of this model lies on the fact that a trace can be seen either as a labelled ordered graph expressing directly the partial order or as an equivalence class of sequences, each of them representing a linearization of the partial order.

In a natural way, and in order to exploit directly the partial order underlying to a trace, a good amount of research have focused on developing temporal logics that can be directly interpreted over traces seen as labelled partially ordered graphs (rather than as sets of sequences). Recently, Thiagarajan and

Walukiewicz [18] have defined a new temporal logic, denoted by LTrL, patterned on the propositional temporal logic of linear time LTL defined by Pnueli [12] with exactly the expressive power of $FO(<)$, the first order theory of finite and infinite traces (see also [10]). This work was the outcome of a long sequence of papers on logics on traces [17,5,1,13].

One of the most important property of LTL is to have a PSPACE-complete decision procedure for LTL is [15] (whereas the decision procedure for the equivalent logic $FO(<)$ is non-elementary [16]). This logic LTL has thus been used successfully in model-checking, see e.g. [3]. So very naturally, people were interested in the complexity of the decision procedure for LTrL which was expected to be of "low" complexity. These hopes vanish after the very recent work of Walukiewicz who shows that the decision procedure for LTrL is non-elementary [19].

This result could be seen as an irremediable drawback for the use of LTrL for "practical" model-checking. Nevertheless, recent tools such as Mona [8] or Mosel [7] have been proposed to handle non-elementary logics. Moreover, significant "real" problems have been solved using these tools. Therefore, it appears worthwhile to have a decision procedure for LTrL. Up to now, the only existing one lies on the transformation of a LTrL-formula into an equivalent $FO(<)$-formula used by Thiagarajan and Walukiewicz to prove their main theorem [18].

To achieve this goal, we propose as the main result of this paper, a modular direct decision procedure for LTrL. Precisely we construct for any LTrL-formula α, a Büchi automaton recognizing the set of (the linearizations of) the models of α. From the well-known decidability of the emptiness of the language recognized by a Büchi automaton, we get our decision procedure. Our construction is performed in a modular way from the structure of α.

In fact, since the logic LTrL is not pure future but contains some "present" operators, we need to construct by induction not a simple automaton but a sequence of automata indexed by some alphabetic information. The constructions for the boolean operators and the local next operators are classical and do not present any new difficulty (note nevertheless that the negation requires of course a complementation of a Büchi automaton). The crucial step is the construction for the "Until" operator. From [19], we know that the operator is responsible of the non-elementariness and therefore the construction can not be simple. We use the notion of alphabetic automaton (classical from finite automata theory) in order to simplify the presentation as much as possible.

Our paper is organized as follows. In Section 2 we set the basic definitions for the trace theory and the temporal logic LTrL and we introduce the problem of Model-Checking in this logic. Section 3 is devoted to the presentation of the main tools we will be using: Büchi automata and alphabetic automata. These tools allow us to describe our modular constructions in Section 4 in a more concise manner. Finally, we give some conclusions about those constructions in Section 5, and devise some possible paths for future research.

2 Model-Checking on LTrL

2.1 Traces

A *dependence alphabet* is a pair (Σ, I) where Σ is a finite set and $I \subseteq \Sigma \times \Sigma$ is a symmetric and irreflexive relation called the *independence relation*. Elements of Σ are called *actions*; two actions $a, b \in \Sigma$ are said *independent* if $(a, b) \in I$. The complementary relation $D = \Sigma \times \Sigma \setminus I$ is called the *dependence relation*. If $a \in \Sigma$, then $I(a)$ and $D(a)$ denote respectively the sets of letters independent and dependent with a.

A (Mazurkiewicz) *trace* on (Σ, I) is a Σ-labelled partially ordered set (poset) that respects the dependence relation [9]. More formally, let $(E, \leq, \lambda)$ be a Σ-labelled poset, that is: E is a finite or infinite set, $\leq$ is a partial order on E and λ is a labelling function from E into Σ. For every subset Y of E, we define $\downarrow Y = \{x \in E \mid \exists y \in Y, x \leq y\}$. If Y is a singleton $\{y\}$ we shall write $\downarrow y$ instead of $\downarrow \{y\}$. We also define the *covering relation* $\lessdot$ on $E \times E$: $x \lessdot y$ if $x < y$ and $\forall z \in E$, $x \leq z \leq y$ implies $z = x$ or $z = y$.

Then a *trace* over (Σ, I) is a Σ-labelled poset $u = (E, \leq, \lambda)$ satisfying:

$$
\begin{aligned}
&(T1)\ \ \forall e \in E,\ \ \downarrow e \text{ is a finite set}\\
&(T2)\ \forall e, e' \in E,\ e \lessdot e' \Rightarrow (\lambda(e), \lambda(e')) \in D\\
&(T3)\ \forall e, e' \in E,\ (\lambda(e), \lambda(e')) \in D \Rightarrow e \leq e' \text{ ou } e' \leq e
\end{aligned}
$$

Elements of E are called *events*. A *configuration* of a trace $u = (E, \leq, \lambda)$ is a finite subset $c \subseteq E$ satisfying $\downarrow c = c$. It can be viewed as a finite trace which is a prefix of u. The set of letters of *maximal* events in c (for the partial order induced by u) is denoted by $max(c)$. We denote by C_u the set of configurations of a trace u. Remark that $\emptyset \in C_u$ and $max(\emptyset) = \emptyset$. The *transition relation* $\rightarrow_u \subseteq C_u \times \Sigma \times C_u$ is given by: $c \xrightarrow{a}_u c'$ iff there exists $e \in E$ such that $\lambda(e) = a$, $e \notin c$ and $c' = c \cup \{e\}$.

2.2 LTrL

Thiagarajan and Walukiewicz[18] have proposed a temporal logic on (finite and infinite) traces, called LTrL, that is expressively complete, i.e. equivalent to the first-order logic $FO(<)$ on (finite and infinite) traces.

The set of formulas belonging to LTrL is defined inductively by: [1]

$$\mathrm{LTrL}(\Sigma, I) ::= \underline{tt} \mid \neg\alpha \mid \alpha \vee \beta \mid @\alpha \mid \alpha\, \mathcal{U}\, \beta \mid \underline{a}$$

The semantics of these formulas are defined inductively: let $u = (E, \leq, \lambda)$ be a trace and c a configuration of u, then

- $u, c \models \underline{tt}$.
  Furthermore, the boolean connectives $\neg$ and $\vee$ have the usual interpretations.

[1] Our notations are slightly different from the ones of the original article.

- $u, c \models @\alpha$ iff $\exists c' \in C_u$, $c \xrightarrow{a}_u c'$ and $u, c' \models \alpha$.
- $u, c \models \alpha \; \mathcal{U} \; \beta$ iff $\exists c' \in C_u$, $c \subseteq c'$, such that $u, c' \models \beta$ and $\forall c'' \in C_u$, $c \subseteq c'' \subset c'$ implies $u, c'' \models \alpha$.
- $u, c \models \underline{a}$ iff $\exists c' \in C_u$ such that $c' \xrightarrow{a}_u c$, i.e. $a \in max(c)$.

Note that the formulas of the form $\underline{a}$ describe the present configuration at which they are evaluated. For this reason, they will be referred to as *present formulas*. Thus, if c and c' are two configurations such that $max(c) = max(c')$, then for any trace u and any formula α of LTrL it holds:

$$cu, c \models \alpha \Leftrightarrow c'u, c' \models \alpha$$

Denote by $\mathbf{R}(\Sigma, I)$ the set of finite and infinite traces on the dependence alphabet (Σ, I), then every formula α of LTrL defines a trace language $L(\alpha)$ in the following way:

$$L(\alpha) = \{u \in \mathbf{R}(\Sigma, I) \mid u, \emptyset \models \alpha\}$$

2.3 The Model-Checking problem

The *global* nature of LTrL's temporal operators makes it easy to specify global liveness and safety properties. However, this global nature has a counterpart regarding the complexity of verification tasks such as satisfiability or model-checking. Actually, Walukiewicz has shown[19] that the satisfiability problem is non-elementary in LTrL, and that this non-elementarity is due to the possible nesting of "until" operators. However, this should not prevent us from looking for direct algorithms for verification tasks. Indeed, such algorithms would allow to check at least formulas with a low "until depth".

Our approach is to start with a formula α of LTrL and build an automaton recognizing the language defined by α inductively on the structure of that formula. With these constructions, the satisfiability and model-checking problems are reduced to checking for emptiness of automata.

Our main result can be stated as follows:

Theorem 1. *Let α be a formula of the temporal logic* LTrL *for a given dependence alphabet (Σ, I). A Büchi automaton $\mathcal{A}_\alpha$ on Σ^∞ that accepts exactly the linearizations of traces satisfying α, can be constructed inductively on the structure of α.*

3 Tools

3.1 Words and Traces

Although we want to recognize *trace* languages, our approach is to construct automata that will recognize I-closed sets of words. More formally, let Σ^∞ be the set of finite and infinite words on the alphabet Σ, and recall that $\mathbf{R}(\Sigma, I)$ is the set of finite and infinite traces on (Σ, I). Now let φ denote the canonical morphism from Σ^∞ into $\mathbf{R}(\Sigma, I)$. A word language $L \subseteq \Sigma^\infty$ is said to be I-closed if it is closed by the morphism φ, i.e. if $L = \varphi^{-1}(\varphi(L))$.

Note that $\mathbf{R}(\Sigma, I)$ can also be defined as $\Sigma^\infty / \sim_I$, where $\sim_I$ is the congruence induced by I on Σ^∞ (see e.g. [4] for further details).

3.2 Büchi Automata

In the sequel, we shall use some *Büchi automata*, that is non-deterministic automata on words with a particular *repeating* acceptance condition for infinite words [2,11].

Definition A Büchi automaton on Σ is a t-uple $\mathcal{A} = (Q, \rightarrow, S, F, R)$ where:

- Q is the finite set of *states*
- $\rightarrow \subseteq Q \times \Sigma \times Q$ the *transition relation.* If $(q, a, q') \in \rightarrow$, we shall write $q \xrightarrow{a} q'$. The transitive closure of $\rightarrow$ will be denoted by $\rightarrow^*$.
- $S \subseteq Q$ is the set of *initial* states of $\mathcal{A}$.
- F is the set of *final* states.
- $R \subseteq Q$ is the set of *repeated* states.

Executions A finite *execution* of $\mathcal{A}$ on a finite word $u = a_1 a_2 \ldots a_n$, with $\forall i, a_i \in \Sigma$ is a sequence $\rho = q_0, q_1, q_2, \ldots, q_n$ of states such that $q_0 \in S$ and for all $0 \leq i \leq n-1$, we have $q_i \xrightarrow{a_{i+1}} q_{i+1}$. If $u = a_1 a_2 \ldots a_k \ldots$ is an infinite word, an *execution* of $\mathcal{A}$ on u is an infinite sequence $\rho = q_0, q_1, q_2, \ldots, q_k, \ldots$ of states such that $q_0 \in S$ and for all $i \leq 0$, it holds $q_i \xrightarrow{a_{i+1}} q_{i+1}$. Denote by $\mathrm{rep}(\rho)$ the set of states $q \in Q$ such that $\{i \geq 0 \mid q = q_i\}$ is infinite. Note that since $\mathcal{A}$ is non-deterministic, it can have different executions on the same word.

Acceptance conditions If u is a finite word, an execution ρ of $\mathcal{A}$ on u is *accepting* if the last state of ρ is in F. If u is a infinite word, an execution ρ of $\mathcal{A}$ on u is *accepting* if $\mathrm{rep}(\rho) \cap R \neq \emptyset$.

Finally, the language $L(\mathcal{A})$ of words *accepted* or *recognized* by $\mathcal{A}$ is the set of (finite and infinite) words w such that there exists an accepting execution of $\mathcal{A}$ on w.

3.3 Alphabetic Automata

For our constructions we will need the notion of *alphabetic* automata: an automaton is alphabetic if all the words that can be used to reach a particular state have the same alphabet.

Formally, a Büchi automaton $\mathcal{A} = (Q, \rightarrow, S, F, R)$ is *alphabetic* if it satisfies the following property:

$$\forall q \in Q, \exists B \subseteq \Sigma, \forall q_0 \in S, \forall u \in \Sigma^*, q_0 \xrightarrow{u}^* q \Rightarrow alph(u) = B$$

If this is indeed the case, then we set $alph(q) = B$.

Our constructions use intensively the following result which can be considered as "folklore" in automata theory:

Proposition 2. *Every Büchi automaton can be transformed into an alphabetic Büchi automaton recognizing the same language.*

Proof. Let $\mathcal{A} = (Q, \rightarrow, S, F, R)$ be a Büchi automaton on the alphabet Σ. Let $\mathcal{B} = (Q_1, \rightarrow_1, S_1, F_1, R_1)$ such that:

- $Q_1 = Q \times 2^{\Sigma}$,
- $(q, E) \xrightarrow{a}_1 (q', E')$ iff $q \xrightarrow{a} q'$ and $E' = E \cup \{a\}$,
- $S_1 = S \times \{\emptyset\}$,
- $F_1 = \bigcup_{G \subseteq \Sigma} F \times G$,
- and $R_1 = \bigcup_{G \subseteq \Sigma} R \times G$.

Then a word u is accepted by $\mathcal{A}$ iff it is accepted by $\mathcal{B}$. Moreover, $\mathcal{B}$ is alphabetic, since if a state (q, E) is reachable from an initial state $(q_0, \emptyset)$ in $\mathcal{B}$ through a word u, then $E = alph(u)$.

4 Modular constructions

4.1 Methodology

If α is an LTrL formula, then $L(\alpha)$ is a *trace* language. Our goal is to build a Büchi automaton on words that recognizes the word language $\varphi^{-1}(L(\alpha))$.We achieve this construction in a modular way, inductively on the structure of the formula α.

One of the problems we encounter while attempting to make such constructions comes from the fact that the logic LTrL is not *pure future-oriented*, since it features present formulas. For instance, if $\alpha = \underline{a}$, then $L(\alpha) = \emptyset$, but $L(@\ \underline{a}) = a\mathbf{R}(\Sigma, I)$. This example shows that $L(\alpha)$ is not a sufficient information for our modular approach. In order to overcome this problem, we shall define, for every formula $\alpha \in$ LTrL a finite family of automata $(\mathcal{A}_{\alpha}^{B})_B$ for every set B *compatible* with I, that is such that $B \times B \subseteq I$. For a *compatible* set B, the automata $\mathcal{A}_{\alpha}^{B}$ will recognize the word language $\varphi^{-1}(L_B(\alpha))$ where $L_B(\alpha)$ is the trace language $\{u \mid \forall c, max(c) = B \Rightarrow cu, c \models \alpha\}$. Intuitively, our idea is that $L_B(\alpha)$ will be the set of traces u that satisfy α starting at a configuration whose maximal letters are described by B. This trick will allow us to deal with $\underline{a}$ formulas more smoothly.

With these notations, the word language $\varphi^{-1}(L(\alpha))$ will be recognized by the automaton $\mathcal{A}_{\alpha}^{\emptyset}$.

Subsection 4.2 deals with the easy cases for the formula α, while the difficult case of formulas of the form $\beta\, \mathcal{U}\, \gamma$ is presented in Subsection 4.3.

4.2 Boolean connectors and next-step operator

The constant formula Assume $\alpha = \underline{tt}$. Then $L(\alpha) = \mathbf{R}(\Sigma, I)$, so $\varphi^{-1}(L(\alpha)) = \Sigma^\infty$. Moreover, for every compatible subset $B \subseteq \Sigma$, the langage $L_B(\alpha)$ is $\mathbf{R}(\Sigma, I)$, so $\varphi^{-1}(L_B(\alpha)) = \Sigma^\infty$. We set $\mathcal{A}^B_{\underline{tt}} = (Q, \rightarrow, S, F, R)$ where:

- $Q = \{0\}$,
- $0 \xrightarrow{a} 0$ for all $a \in \Sigma$,
- $S = \{0\}$,
- and $F = R = \{0\}$.

In that way, it is obvious that every (finite or infinite) word is accepted by $\mathcal{A}^B_{\underline{tt}}$.

The present formulas Assume that $\alpha = \underline{a}$ for some $a \in \Sigma$. Now, depending on the set B we choose, we will construct two different automata. Indeed, if $a \in B$, then $L_B(\underline{a}) = \mathbf{R}(\Sigma, I)$, while if $a \notin B$, then $L_B(\underline{a}) = \emptyset$.

- *For* $B \ni a$, we define $\mathcal{A}^B_\alpha = \mathcal{A}^B_{\underline{tt}}$;
- *For* $B \not\ni a$, we define $\mathcal{A}^B_\alpha = \mathcal{A}^B_{\neg\underline{tt}}$, this automaton being the same as $\mathcal{A}^B_{\underline{tt}}$ except that $F = R = \emptyset$.

In both cases, the automaton we construct recognizes exactly $\varphi^{-1}(L_B(\alpha))$.

Negation Assume that $\alpha = \neg\beta$, and that the family of automata $(\mathcal{A}^B_\beta)_B$ is given. For a fixed B, the automaton $\mathcal{A}^B_\alpha$ recognizing exactly the language $\Sigma^\infty \setminus \varphi^{-1}(L_B(\beta))$, which is precisely $\varphi^{-1}(L_B(\alpha))$, can be obtained by the classical method of Safra[14]. Note that this step involves an exponential blow-up in the size of the automata.

Disjunction Assume that we know the automata $\mathcal{A}^B_\beta$ and $\mathcal{A}^B_\gamma$ recognizing respectively the languages $\varphi^{-1}(L_B(\beta))$ and $\varphi^{-1}(L_B(\gamma))$.

We want to construct the automata $(\mathcal{A}^B_\alpha)_B$ for the formula $\alpha = \beta \vee \gamma$. We can assume that $\mathcal{A}^B_\beta$ and $\mathcal{A}^B_\gamma$ have disjoint sets of states, and define $\mathcal{A}^B_\alpha$ as the disjoint union of these two automata. It is clear that $\mathcal{A}^B_\alpha$ recognizes $\varphi^{-1}(L_B(\beta)) \cup \varphi^{-1}(L_B(\gamma))$, i.e. $\varphi^{-1}(L_B(\beta) \cup L_B(\gamma))$, which is precisely $\varphi^{-1}(L_B(\alpha))$.

The indexed next-step operator Assume that $\alpha = ⓐ\beta$ for some $a \in \Sigma$ and some $\beta \in$ LTrL. Assume that we have a fixed compatible set B. Then it is easy to see that $L_B(ⓐ\beta) = aL_{B'}(\beta)$ with $B' = (B \cap I(a)) \cup \{a\}$.

Therefore we define the automaton $\mathcal{A}^B_\alpha = (Q_1, \rightarrow_1, S_1, F_1, R_1)$ from the automaton $\mathcal{A}^{B'}_\beta = (Q, \rightarrow, S, F, R)$ (with $B' = (B \cap I(a)) \cup \{a\}$ as above) as follows:

- $Q_1 = Q \times \{0, 1, 2\}$,
- $S_1 = S \times \{0\}$,

- $F_1 = F \times \{1\}$,
- $R_1 = R \times \{1\}$,

and $\rightarrow_1$ in the following way:

- $(p, 0) \xrightarrow{b}_1 (p', 0)$ iff $b \in I(a)$ and $p \xrightarrow{b} p'$;
- $(p, 0) \xrightarrow{b}_1 (p', 2)$ iff $b \in D(a) \setminus \{a\}$ and $p \xrightarrow{b} p'$;
- $(p, 0) \xrightarrow{b}_1 (p', 1)$ iff $b = a$ and $p \xrightarrow{b} p'$;
- $(p, 1) \xrightarrow{b}_1 (p', 1)$ iff $p \xrightarrow{b} p'$;
- $(p, 2) \xrightarrow{b}_1 (p', 2)$ iff $p \xrightarrow{b} p'$.

It should be clear that $\mathcal{A}_\alpha^B$ recognizes exactly $\varphi^{-1}(L_B(\alpha))$ from the preceding remark.

4.3 A construction for the Until operator

We first explain the idea of the simple construction in the word case. If we want to check some property of the form $\beta \, \mathcal{U} \, \gamma$ on some word w, each time we read a letter a we have to start a new verification for β, until some time in the future when we will start to check the property γ. So every new verification started for β will be added to some set P, but the size of this set is bounded since if two different verifications reach the same state $p \in P$, then we know that we only have to carry on with one verification.

The situation with traces is much more involved. We also have to check β for all suffixes up to the suffix which satisfies γ. The problem is that when a trace t is represented by a word w, the suffixes of t are represented by certain subwords of w and not only suffixes of w. Therefore, each time we read a letter a we might either skip it or apply it to the current verification of β and γ depending on whether or not it is part of the considered trace suffix. Moreover, when we start checking γ, we might still need to start new verifications of β.

Finally, since we deal with infinite computation we need to take care of the acceptance conditions. Since the states that code the various verifications of β are put together in a set structure, we cannot know directly what are the repeated states of each verification. To overcome this problem, we will take an *ordering* transition that will transform this set into an ordered one, thus allowing to trace each verification individually. Note that this problem is not specific to traces.

Notations Assume that $\alpha = \beta \, \mathcal{U} \, \gamma$, and that the families of automata $(\mathcal{A}_\beta^B)_B$ and $(\mathcal{A}_\gamma^B)_B$ are given. We set $\mathcal{A}_\beta^B = (Q_1^B, \rightarrow_1^B, S_1^B, F_1^B, R_1^B)$ and $\mathcal{A}_\gamma^B = (Q_2^B, \rightarrow_2^B, S_2^B, F_2^B, R_2^B)$. We can assume, without loss of generality, that the sets $(Q_i^B)_B$ are pairwise disjoint for a fixed $i = 1, 2$. Now let $\mathcal{A}_1$ (resp. $\mathcal{A}_2$) be the disjoint union of the automata $\mathcal{A}_\beta^B$ (resp. $\mathcal{A}_\gamma^B$) for every compatible set $B \subseteq \Sigma$.

From Proposition 2, we can assume without loss of generality that $\mathcal{A}_1$ and $\mathcal{A}_2$ are alphabetic. We set, for $i = 1, 2$, $\mathcal{A}_i = (Q_i, \rightarrow_i, S_i, F_i, R_i)$.

Let's fix some total order on the set Q_1. This order induces a function $f : 2^{Q_1} \rightarrow Q_1^{\leq n}$ with $n = \mid Q_1 \mid$, which maps every subset P of Q_1 to its corresponding ordered set $\underline{P}$.

Set of states and transitions Recall that $\alpha = \beta\,\mathcal{U}\,\gamma$. For a fixed compatible set B, we define $\mathcal{A}_\alpha^B = (Q, \rightarrow, S, F, R)$ as follows:

- $Q = (2^{Q_1} \times Q_2 \times 2^{\Sigma}) \cup (Q_1^{\leq n} \times Q_2 \times 2^{\Sigma})$,
- $S = \{\emptyset\} \times S_2 \times \{B\}$,
- $F = 2^{F_1} \times F_2 \times 2^{\Sigma}$,
- $R = R_1^{\leq n} \times R_2 \times 2^{\Sigma}$,

and let the transition relation $\rightarrow$ be defined in the following way:

- $(P, q, E) \xrightarrow{a} (P', q, E')$ iff $alph(q) \subseteq I(a)$ and $P' = P_a \cup (P \cap I(a)) \cup \{q_E\}$ such that $\exists q \in S_1^E, q \xrightarrow{a}_1 q_E$; and P_a satisfies $\forall q \in P, \exists q' \in P_a, q \xrightarrow{a}_1 q'$ and $\forall q' \in P_a, \exists q \in P, q \xrightarrow{a}_1 q'$; and $P \cap I(a) = \{q \in P \mid alph(q) \subseteq I(a)\}$; and finally $E' = (E \cap I(a)) \cup \{a\}$. These transitions are called β-transitions.
- $(P, q, E) \xrightarrow{a} (P', q', E')$ iff $P' = P_a$, $q \xrightarrow{a}_2 q'$, and $E' = (E \cap I(a)) \cup \{a\}$. These transitions are called γ-transitions.
- $(P, q, E) \xrightarrow{a} (\underline{P}, q', E')$ iff $\underline{P} = f(P_a)$, $q \xrightarrow{a}_2 q'$ and $E' = (E \cap I(a)) \cup \{a\}$. These transitions are called *ordering* transitions.
- $(\underline{P}, q, E) \xrightarrow{a} (\underline{P'}, q', E')$ iff $\exists n_0 \leq n \mid \underline{P} \in Q_1^{n_0}$ and $\underline{P'} \in Q_1^{n_0}$ and for all $1 \leq i \leq n_0$, $\underline{P}_i \xrightarrow{a}_1 \underline{P'}_i$; $q \xrightarrow{a}_2 q'$; and $E' = (E \cap I(a)) \cup \{a\}$. These transitions are called *finishing* transitions.

Proposition 3. *The automaton $\mathcal{A}_\alpha^B$ recognizes exactly the word language $\varphi^{-1}(L_B(\beta\,\mathcal{U}\,\gamma))$.*

Due to lack of space, we do not provide here a proof of this proposition. The interested reader shall find the complete proof in a technical report[6].

5 Conclusion

We have shown that it is possible to have a direct construction of a Büchi automaton for every formula of LTrL and to make these constructions in a modular way, which allows us to reuse the automata for a formula α in any construction for a formula β where α would be a subformula of β. The two constructions involving an exponential blow-up are for the negation and the until operator. We know that the blow-up for the until operator is unavoidable. As far as the negation is concerned, we could use some transformations on the formulas in order to push every negation inside as far as possible, and describe an easy construction for a conjunction operator. What remains to be done is to find efficient algorithms for fragments of LTrL that are elementary, as Walukiewicz has open the way in [19], and find good characterizations of the expressive power of these fragments.

References

1. R. Alur, D. Peled, and W. Penczek. Model-checking of causality properties. In *Proceedings of LICS'95*, pages 90–100, 1995.
2. J.R. Büchi. Weak second-order arithmetic and finite automata. *Z. Math Logik Grundlag. Math.*, 6:66–92, 1960.
3. C. Courcoubetis, M. Y. Vardi, P. Wolper, and M. Yannakakis. Memory efficient algorithms for the verification of temporal properties. *formal Methods in System Design*, 1:275–288, 1992.
4. V. Diekert and G. Rozenberg, editors. *The Book of Traces.* World Scientific, Singapore, 1995.
5. W. Ebinger. *Charakterisierung von Sprachklassen unendlicher Spuren durch Logiken.* Dissertation, Institut für Informatik, Universität Stuttgart, 1994.
6. P. Gastin, R. Meyer, and A. Petit. A (non-elementary) modular decision procedure for LTrL. Technical report, LSV, ENS de Cachan, June 1998.
7. P. Kelb, T. Margaria, M. Mendler, and C. Gsottberger. Mosel: a flexible toolset for monadic second-order logic. In *Proceedings of CAV'97, LNCS 1254*, 1997.
8. N. Klarlund. Mona & Fido: The logic-automaton connection in practice. In *Proceedings of CSL'97, LNCS*, 1998.
9. A. Mazurkiewicz. Concurrent program schemes and their interpretations. DAIMI Rep. PB 78, Aarhus University, Aarhus, 1977.
10. R. Meyer and A. Petit. Expressive completeness of LTrL on finite traces: an algebraic proof. In *Proceedings of STACS'98*, number 1373 in LNCS, pages 533–543, 1998.
11. D. Perrin and J. E. Pin. Infinite words. Technical report, LITP, Avril 1997.
12. A. Pnueli. The temporal logics of programs. In *Proceedings of the 18th IEEE FOCS, 1977*, pages 46–57, 1977.
13. R. Ramanujam. Locally linear time temporal logic. In *Proceedings of LICS'96*, pages 118–128, 1996.
14. S. Safra. On the complexity of ω-automata. In *Proceedings of the 29th annual IEEE Symp. on Foundations of Computer Science*, pages 319–327, 1988.
15. A. Sistla and E. Clarke. The complexity of propositional linear time logic. *J. ACM*, 32:733–749, 1985.
16. L. Stockmeyer. The complexity of decision problems in automata theory and logic. PhD thesis, TR 133, M.I.T., Cambridge, 1974.
17. P. S. Thiagarajan. A trace based extension of linear time temporal logic. In *Proceedings of the 9th Annual IEEE Symposium on Logic in Computer Science (LICS'94)*, pages 438–447, 1994.
18. P. S. Thiagarajan and I. Walukiewicz. An expressively complete linear time temporal logic for Mazurkiewicz traces. In *Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science (LICS'97)*, 1997.
19. I. Walukiewicz. Difficult configurations - on the complexity of LTrL. In *Proceedings of ICALP'98*, 1998.