BRICS

Basic Research in Computer Science

A Product Version of Dynamic Linear Time Temporal Logic

Jesper G. Henriksen
P. S. Thiagarajan

BRICS Report Series RS-97-9
ISSN 0909-0878 April 1997

BRICS RS-97-9 Henriksen & Thiagarajan: A Product Version of Dynamic Linear Time Temporal Logic

See back inner page for a list of recent BRICS Report Series publications. Copies may be obtained by contacting:

BRICS
Department of Computer Science
University of Aarhus
Ny Munkegade, building 540
DK–8000 Aarhus C
Denmark

Telephone: +45 8942 3360
Telefax: +45 8942 3255
Internet: BRICS@brics.dk

BRICS publications are in general accessible through the World Wide Web and anonymous FTP through these URLs:

`http://www.brics.dk`
`ftp://ftp.brics.dk`
This document in subdirectory `RS/97/9/`

A Product Version of Dynamic Linear Time Temporal Logic*

Jesper Gulmann Henriksen
BRICS[†] Department of Computer Science,
University of Aarhus, Denmark
email: gulmann@brics.dk

P. S. Thiagarajan[‡]
SPIC Mathematical Institute, Madras, India
email: pst@smi.ernet.in

April 1997

Abstract

We present here a linear time temporal logic which simultaneously extends LTL, the propositional temporal logic of linear time, along two dimensions. Firstly, the until operator is strengthened by indexing it with the regular programs of propositional dynamic logic (PDL). Secondly, the core formulas of the logic are decorated with names of sequential agents drawn from fixed finite set. The resulting logic has a natural semantics in terms of the runs of a distributed program consisting of a finite set of sequential programs that communicate by performing common actions together. We show that our logic, denoted $\mathrm{DLTL}^{\otimes}$, admits an exponential time decision procedure. We also show that $\mathrm{DLTL}^{\otimes}$ is expressively equivalent to the so called regular product languages.

*Appears in the proceedings of the eighth International Conference on Concurrency Theory (CONCUR'97), Warsaw, Poland.

[†]**B**asic **R**esearch **i**n **C**omputer **S**cience,
Centre of the Danish National Research Foundation.

[‡]Part of this work was done while visiting BRICS.
Part of this work has been supported by IFCPAR Project 1502-1.

Roughly speaking, this class of languages is obtained by starting with synchronized products of (ω-)regular languages and closing under boolean operations. We also sketch how the behaviours captured by our temporal logic fit into the framework of labelled partial orders known as Mazurkiewicz traces.

1 Introduction

We present a linear time temporal logic which extends LTL, the propositional temporal logic of linear time [8, 13] along two dimensions. Firstly, we strengthen the until modality by indexing it with the regular programs of PDL, propositional dynamic logic [1, 4]. Secondly, we consider networks of sequential agents that communicate by performing common actions together. We then reflect this in the logic by decorating the "core" formulas with the names of the agents. The resulting logic, denoted DLTL$^{\otimes}$, is a smooth generalization of the logic called product LTL [16] and the logic called dynamic linear time temporal logic [5].

DLTL$^{\otimes}$ admits a pleasant theory and our technical goal here is to sketch the main results of this theory. We believe that these results provide additional evidence — in a non-sequential setting — suggesting that our technique of combining dynamic and temporal logic as initiated in [5] is a fruitful one.

In the next section we introduce dynamic linear time temporal logic. We then state two main results concerning this logic. In Section 3 we define regular product languages. These are basically boolean combinations of synchronized products of (ω-)regular languages. We then present a characterization of this class of languages in terms of networks of Büchi automata that coordinate their activities by synchronizing on common letters.

In Section 4 we formulate the temporal logic DLTL$^{\otimes}$, the main object of study in this paper. In Section 5 we establish an exponential time decision procedure for this logic by exploiting the Büchi automata networks presented in Section 3. In the subsequent section we show that DLTL$^{\otimes}$ captures exactly the class of regular product languages. It is worth noting that this is the first temporal logical characterization of this important class of distributed behaviours. In the final section we sketch how the behaviours described by our temporal logic (i.e. regular product languages) lie naturally within the domain of regular Mazurkiewicz trace languages.

2 Dynamic Linear Time Temporal Logic

One key feature of the syntax and semantics of our temporal logic is that *actions* will be treated as first class objects. The usual presentation of LTL [8, 13] is based on *states*; they are represented as subsets of a finite set of atomic propositions. We wish to bring in actions explicitly because it is awkward, if not difficult, to define synchronized products of sequential components in a purely state-based setting. This method of forming distributed systems is a common and useful one. Moreover, it is the main focus of attention in this paper. Hence it will be handy to work with logics in which both states and actions can be treated on an equal footing. As a vehicle for introducing some terminology we shall first introduce an action-based version of LTL denoted LTL(Σ). We begin with some notations.

Through the rest of the paper we fix a finite non-empty alphabet Σ. We let a, b range over Σ and refer to members of Σ as actions. Σ^* is the set of finite words and Σ^ω is the set of infinite words generated by Σ with $\omega = \{0, 1, \ldots\}$. We set $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$ and denote the null word by ε. We let σ, σ' range over Σ^∞ and τ, τ', τ'' range over Σ^*. Finally, $\preceq$ is the usual prefix ordering defined over Σ^* and $\mathrm{prf}(\sigma)$ is the set of finite prefixes of σ.

The set of formulas of LTL(Σ) is then given by the syntax:

$$\mathrm{LTL}(\Sigma) ::= \top \mid \sim\alpha \mid \alpha \vee \beta \mid \langle a \rangle \alpha \mid \alpha \,\mathcal{U}\, \beta.$$

For convenience we have avoided introducing atomic propositions and instead just deal with the constant $\top$ and its negation $\sim\top \stackrel{\Delta}{\Longleftrightarrow} \bot$. Through the rest of this section α, β will range over LTL(Σ). The modality $\langle a \rangle$ is an action-indexed version of the next-state modality of LTL. A model is a ω-sequence $\sigma \in \Sigma^\omega$. For $\tau \in \mathrm{prf}(\sigma)$ we define $\sigma, \tau \models \alpha$ via:

- $\sigma, \tau \models \top$.
- $\sigma, \tau \models \sim\alpha$ iff $\sigma, \tau \not\models \alpha$.
- $\sigma, \tau \models \alpha \vee \beta$ iff $\sigma, \tau \models \alpha$ or $\sigma, \tau \models \beta$.
- $\sigma, \tau \models \langle a \rangle \alpha$ iff $\tau a \in \mathrm{prf}(\sigma)$ and $\sigma, \tau a \models \alpha$.
- $\sigma, \tau \models \alpha \,\mathcal{U}\, \beta$ iff there exists τ' such that $\tau\tau' \in \mathrm{prf}(\sigma)$ and $\sigma, \tau\tau' \models \beta$. Further, for every τ'' such that $\varepsilon \preceq \tau'' \prec \tau'$, it is the case that $\sigma, \tau\tau'' \models \alpha$.

It is well known that LTL(Σ) is equal in expressive power to the first order theory of sequences [2, 7]. Consequently this temporal logic is quite limited in in terms of what it can not say. As an example, let $\Sigma = \{a, b\}$. Then the property "at every even position the action b is executed" is not definable in LTL(Σ). This observation, made in a state-based setting by Wolper, is the starting point for the extension of LTL called ETL [20, 21]. The route that we have taken to augment the expressive power of LTL(Σ) is similar in spirit but quite different in terms of the structuring mechanisms made available for constructing compound formulas. A more detailed assessment of the similarities and the differences between the two approaches is given in [5].

The extension that we have proposed is called DLTL(Σ). It basically consists of indexing the until operator with the programs of PDL (e.g. [1]). We start by defining the set of regular programs (expressions) generated by Σ. This set is denoted by Prg(Σ) and its syntax is given by:

$$\mathrm{Prg}(\Sigma) ::= a \mid \pi_0 + \pi_1 \mid \pi_0; \pi_1 \mid \pi^*.$$

With each program we associate a set of finite words via the map $||\cdot|| : \mathrm{Prg}(\Sigma) \longrightarrow 2^{\Sigma^*}$. This map is defined in the standard fashion:

- $||a|| = \{a\}$.
- $||\pi_0 + \pi_1|| = ||\pi_0|| \cup ||\pi_1||$.
- $||\pi_0; \pi_1|| = \{\tau_0\tau_1 \mid \tau_0 \in ||\pi_0|| \text{ and } \tau_1 \in ||\pi_1||\}$.
- $||\pi^*|| = \bigcup_{i\in\omega} ||\pi^i||$, where
  - $||\pi^0|| = \{\varepsilon\}$ and
  - $||\pi^{i+1}|| = \{\tau_0\tau_1 \mid \tau_0 \in ||\pi|| \text{ and } \tau_1 \in ||\pi^i||\}$ for every $i \in \omega$.

The set of formulas of DLTL(Σ) is given by the following syntax.

$$\mathrm{DLTL}(\Sigma) ::= \top \mid \sim\alpha \mid \alpha \vee \beta \mid \alpha\, \mathcal{U}^{\pi} \beta, \quad \pi \in \mathrm{Prg}(\Sigma)$$

A model is a ω-sequence $\sigma \in \Sigma^\omega$. For $\tau \in \mathrm{prf}(\sigma)$ we define $\sigma, \tau \models \alpha$ just as we did for LTL(Σ) in the case of the first three clauses. As for the last one,

- $\sigma, \tau \models \alpha\, \mathcal{U}^{\pi} \beta$ iff there exists $\tau' \in ||\pi||$ such that $\tau\tau' \in \mathrm{prf}(\sigma)$ and $\sigma, \tau\tau' \models \beta$. Moreover, for every τ'' such that $\varepsilon \preceq \tau'' \prec \tau'$, it is the case that $\sigma, \tau\tau'' \models \alpha$.

Thus DLTL(Σ) adds to LTL(Σ) by strengthening the until operator. To satisfy $\alpha\ \mathcal{U}^{\pi}\beta$, one must satisfy $\alpha\mathcal{U}\beta$ along some finite stretch of behaviour which is required to be in the (linear time) behaviour of the program π. We now wish to state two of the main results of [5]. To do so, we first say that a formula $\alpha \in$ DLTL(Σ) is *satisfiable* if there exist $\sigma \in \Sigma^{\omega}$ and $\tau \in \mathrm{prf}(\sigma)$ such that $\sigma, \tau \models \alpha$. Secondly, we associate with a formula α the ω-language L_{α} via:

$$L_{\alpha} \stackrel{\mathrm{def}}{=} \{\sigma \in \Sigma^{\omega} \mid \sigma, \varepsilon \models \alpha\}.$$

A language $L \subseteq \Sigma^{\omega}$ is said to be DLTL(Σ)-definable if there exists some $\alpha \in$ DLTL(Σ) such that $L = L_{\alpha}$. Finally, we assume the notions of Büchi and Muller automata and ω-regular languages as formulated in [17].

Theorem 2.1

(i) *Given an* $\alpha_0 \in$ DLTL(Σ) *one can effectively construct a Büchi automaton* $\mathcal{B}_{\alpha_0}$ *of size* $2^{O(|\alpha_0|)}$ *such that* $\mathcal{L}(\mathcal{B}_{\alpha_0}) \neq \emptyset$ *iff* α_0 *is satisfiable. Thus the satisfiability problem for* DLTL(Σ) *is decidable in exponential time.*

(ii) $L \subseteq \Sigma^{\omega}$ *is* ω*-regular iff* L *is* DLTL(Σ)*-definable.*

It is also easy to formulate and solve a natural model checking problem for DLTL(Σ) where finite state programs are modelled as Büchi automata. But we shall not enter into details here.

To close out the section we shall point to two useful derived operators of DLTL(Σ):

- $\langle\pi\rangle\alpha \stackrel{\Delta}{\Longleftrightarrow} \top\ \mathcal{U}^{\pi}\alpha$.
- $[\pi]\alpha \stackrel{\Delta}{\Longleftrightarrow} \sim\langle\pi\rangle\sim\alpha$.

Suppose $\sigma \in \Sigma^{\omega}$ is a model and $\tau \in \mathrm{prf}(\sigma)$. It is easy to see that $\sigma, \tau \models \langle\pi\rangle\alpha$ iff there exists $\tau' \in ||\pi||$ such that $\tau\tau' \in \mathrm{prf}(\sigma)$ and $\sigma, \tau\tau' \models \alpha$. It is also easy to see that $\sigma, \tau \models [\pi]\alpha$ iff for every $\tau' \in ||\pi||$, $\tau\tau' \in \mathrm{prf}(\sigma)$ implies $\sigma, \tau\tau' \models \alpha$. In this sense, the program modalities of PDL acquire a linear time semantics in the present setting. As shown in [5] the second part of Theorem 2.1 goes through even for the the sublogic of DLTL(Σ) obtained by banishing the until operator and instead using $\langle\pi\rangle\alpha$ and the boolean connectives. For an example of what can be said in this sublogic, assume $\Sigma = \{a, b\}$ and define π_{ev} to be the program $((a + b); (a + b))^{*}$.

Then the formula $[\pi_{ev}]\langle b\rangle\top$ says "at every even position the action b is executed".

Next we note that $a \in \Sigma$ is a member of $\mathrm{Prg}(\Sigma)$ and the until operator of LTL(Σ) can be obtained via: $\alpha\mathcal{U}\beta \stackrel{\Delta}{\Longleftrightarrow} \alpha\mathcal{U}^{\Sigma^*}\beta$. Due to second part of Theorem 2.1 we now have that both syntactically and semantically, LTL(Σ) is a *proper* fragment of DLTL(Σ).

To conclude the section, we note that the material presented here can be easily extended to include finite sequences over Σ as well. We shall assume from now on that this extension has indeed been carried out.

3 Regular Product Languages

A restricted but very useful model of finite state concurrent programs consists of a fixed number of finite state *sequential* programs that coordinate their activities by performing common actions together. A regular product language is an abstract specification of the linear time behaviour of such concurrent programs. The idea is to start with synchronized products of regular languages and close under boolean operations. Formally, we start with a *distributed alphabet* $\widetilde{\Sigma} = \{\Sigma_i\}_{i=1}^{K}$, a family of alphabets with each Σ_i a non-empty finite set of actions. One key point is that the component alphabets are not necessarily disjoint. Intuitively, $Loc = \{1, \ldots, K\}$ is the set of names of communicating sequential processes synchronizing on common actions, where Σ_i is the set of actions which require the participation of the agent i. Through the rest of the paper we fix a distributed alphabet $\widetilde{\Sigma} = \{\Sigma_i\}_{i=1}^{K}$ and set $\Sigma = \bigcup_{i=1}^{K} \Sigma_i$. We carry over the terminology developed in the previous section for dealing with finite and infinite sequences over Σ. In addition, for $\sigma \in \Sigma^\infty$ and $i \in Loc$ we denote by $\sigma \restriction i$ the projection of σ down to Σ_i. In other words, it is the sequence obtained by erasing from σ all occurrences of symbols that are not in Σ_i. We let i, j, k range over $Loc = \{1, \ldots, K\}$ and define $Loc(a) = \{i \mid a \in \Sigma_i\}$. We note that $Loc(a)$ is the set of processes that participate in each occurrence of the action a.

Next we define the K-ary operator $\otimes : 2^{\Sigma_1^\infty} \times \ldots \times 2^{\Sigma_K^\infty} \to 2^{\Sigma^\infty}$ by

$$\otimes(L_1, \ldots, L_K) = \{\sigma \in \Sigma^\infty \mid \sigma \restriction i \in L_i \text{ for each } i \in Loc\}.$$

Usually we will write $\otimes(L_1, \ldots, L_K)$ as $L_1 \otimes L_2 \otimes \ldots \otimes L_K$. Finally, we will say that the language $L \subseteq \Sigma^\infty$ is regular iff $L \cap \Sigma^*$ is a regular subset of Σ^* and $L \cap \Sigma^\omega$ is an ω-regular subset of Σ^ω. Regular product languages can be built up as follows.

Definition 3.1 $L \subseteq \Sigma^\infty$ is a *direct* regular product language over $\widetilde{\Sigma}$ iff $L = L_1 \otimes \ldots \otimes L_K$ with L_i a regular subset of Σ_i^∞ for each $i \in Loc$.

We let $\mathcal{R}_0^\otimes(\widetilde{\Sigma})$ be the class of direct regular product languages over $\widetilde{\Sigma}$.

Definition 3.2 The class of regular product languages over $\widetilde{\Sigma}$ is denoted $\mathcal{R}^\otimes(\widetilde{\Sigma})$ and is the least class of languages containing $\mathcal{R}_0^\otimes(\widetilde{\Sigma})$ and satisfying:

- If $L_1, L_2 \in \mathcal{R}^\otimes(\widetilde{\Sigma})$ then $L_1 \cup L_2 \in \mathcal{R}^\otimes(\widetilde{\Sigma})$.

In what follows we will often suppress the mention of the distributed alphabet $\widetilde{\Sigma}$. It is easy to prove that $\mathcal{R}^\otimes$ is closed under boolean operations. The proof of this result as well as other results mentioned in this section can be found in [15]. Just as ω-regular languages are captured by Büchi automata, we can capture regular product languages with the help of networks of Büchi automata. For convenience such automata will be termed product automata.

Definition 3.3 A *product automaton over* $\widetilde{\Sigma}$ is a structure

$$\mathcal{A} = (\{\mathcal{A}_i\}_{i \in Loc}, Q_{in}),$$

where each $\mathcal{A}_i = (Q_i, \longrightarrow_i, F_i, F_i^\omega)$ satisfies:

- Q_i is a non-empty finite set of i-local states.
- $\longrightarrow_i \subseteq Q_i \times \Sigma_i \times Q_i$ is the transition relation of the ith component.
- $F_i \subseteq Q_i$ is a set of finitary accepting states of the ith component.
- $F_i^\omega \subseteq Q_i$ is a set of infinitary accepting states of the ith component.

Moreover, $Q_{in} \subseteq Q_1 \times \ldots \times Q_K$ is a set of global initial states.

Thus, a product automaton is a network of local automata with a global set of initial states. It is necessary to have global initial states in order to obtain the required expressive power. Each local automaton is equipped to cope with both finite and infinite behaviours using the finitary and infinitary accepting states. The infinitary accepting states are to be interpreted as defining a Büchi acceptance condition. This will become clear once we define the language accepted by a product automaton. We choose to deal with both finite and infinite component behaviours because the global behaviour can always induce finite local behaviours. In other words, even if ω-behaviour is the main focus of interest, a global infinite run will consist of one or more components running forever but with some other components, in general, quitting after making a finite number of moves. The notational complications involved in artificially making *all* components to run forever do not seem to be worth the trouble.

Let $\mathcal{A}$ be a product automaton over $\widetilde{\Sigma}$. Then $Q_G^{\mathcal{A}} = Q_1 \times \ldots \times Q_K$ is the set of global states of $\mathcal{A}$. The i-local transition relations induce a global transition relation $\longrightarrow_{\mathcal{A}} \subseteq Q_G^{\mathcal{A}} \times \Sigma \times Q_G^{\mathcal{A}}$ as follows:

$$q \xrightarrow{a}_{\mathcal{A}} q' \quad \text{iff} \quad \begin{array}{l} q[i] \xrightarrow{a}_i q'[i] \text{ for each } i \in Loc(a) \text{ and} \\ q[i] = q'[i] \text{ for each } i \notin Loc(a), \end{array}$$

where $q[i]$ denotes the ith component of $q = (q_1, \ldots, q_K)$. A run of $\mathcal{A}$ over $\sigma \in \Sigma^\infty$ is a mapping $\rho : \mathrm{prf}(\sigma) \to Q_G^{\mathcal{A}}$ satisfying that $\rho(\varepsilon) \in Q_{in}$ and $\rho(\tau) \xrightarrow{a}_{\mathcal{A}} \rho(\tau a)$ for all $\tau a \in \mathrm{prf}(\sigma)$. The run is *accepting* iff the following conditions are satisfied for each i:

- If $\sigma \upharpoonright i$ is finite then $\rho(\tau)[i] \in F_i$ for some $\tau \in \mathrm{prf}(\sigma)$ with $\tau \upharpoonright i = \sigma \upharpoonright i$.
- If $\sigma \upharpoonright i$ is infinite then $\rho(\tau)[i] \in F_i^\omega$ for infinitely many $\tau \in \mathrm{prf}(\sigma)$.

We next define

$$\mathcal{L}(\mathcal{A}) = \{\sigma \in \Sigma^\infty \mid \text{there exists an accepting run of } \mathcal{A} \text{ over } \sigma\}.$$

The next result established relates regular product languages to product automata.

Theorem 3.4 *$L \in \mathcal{R}^{\otimes}(\widetilde{\Sigma})$ iff $L = \mathcal{L}(\mathcal{A})$ for some product automaton $\mathcal{A}$ over $\widetilde{\Sigma}$.*

We will later give solutions to the satisfiability problem for a product version of DLTL with the help of product automata. The following results will be useful in this context. In stating these results we take the *size* of the product automaton $\mathcal{A}$ to be $|Q_G^{\mathcal{A}}|$.

Lemma 3.5

- *Let $\mathcal{A}$ be a product automaton. The question $\mathcal{L}(\mathcal{A}) \stackrel{?}{\neq} \emptyset$ can be effectively decided in time $O(n^2)$, where n is the size of $\mathcal{A}$.*

- *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be product automata of sizes n_1 and n_2, respectively. Then a product automaton $\mathcal{A}$ of size $O(n_1 n_2)$ with $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$ can be effectively constructed.*

4 A Product Version of DLTL

We now wish to design a product version of DLTL denoted $\mathrm{DLTL}^{\otimes}(\widetilde{\Sigma})$. It will turn out to have the expressive power of regular product languages over $\widetilde{\Sigma}$. The set of formulas and their *locations* are given by:

- $\top$ is a formula and $\mathrm{loc}(\top) = \emptyset$.

- Suppose α and β are formulas. Then so are $\sim\alpha$ and $\alpha \vee \beta$. Furthermore, $\mathrm{loc}(\sim\alpha) = \mathrm{loc}(\alpha)$ and $\mathrm{loc}(\alpha \vee \beta) = \mathrm{loc}(\alpha) \cup \mathrm{loc}(\beta)$.

- Suppose α and β are formulas such that $\mathrm{loc}(\alpha), \mathrm{loc}(\beta) \subseteq \{i\}$ and suppose $\pi \in \mathrm{Prg}(\Sigma_i)$. Then $\alpha\ \mathcal{U}_i^{\pi} \beta$ is a formula. Moreover, $\mathrm{loc}(\alpha\, \mathcal{U}_i^{\pi} \beta) = \{i\}$.

We note that each formula in $\mathrm{DLTL}^{\otimes}(\widetilde{\Sigma})$ is a boolean combination of formulas taken from the set $\bigcup_{i \in Loc} \mathrm{DLTL}_i^{\otimes}(\widetilde{\Sigma})$ where, for each i,

$$\mathrm{DLTL}_i^{\otimes}(\widetilde{\Sigma}) = \{\alpha \mid \alpha \in \mathrm{DLTL}^{\otimes}(\widetilde{\Sigma}) \text{ and } \mathrm{loc}(\alpha) \subseteq \{i\}\ \}.$$

Once again, we have chosen to avoid dealing with atomic propositions for the sake of convenience. They can be introduced in a local fashion as done in [15]. The decidability result to be presented will go through with minor notational overheads.

As before, we will often suppress the mention of $\widetilde{\Sigma}$. We will also often write τ_i, τ_i' and τ_i'' instead of $\tau \restriction i$, $\tau' \restriction i$ and $\tau'' \restriction i$, respectively with $\tau, \tau', \tau'' \in \Sigma^*$.

A model is a sequence $\sigma \in \Sigma^{\infty}$ and the semantics of this logic is given as before, with $\tau \in \mathrm{prf}(\sigma)$.

- $\sigma, \tau \models \top$.

- $\sigma, \tau \models \sim\alpha$ iff $\sigma, \tau \not\models \alpha$.

- $\sigma, \tau \models \alpha \vee \beta$ iff $\sigma, \tau \models \alpha$ or $\sigma, \tau \models \beta$.

- $\sigma, \tau \models \alpha\ \mathcal{U}_i^\pi \beta$ iff there exists τ' such that $\tau'_i \in ||\pi||$ (recall that $\tau'_i = \tau' \restriction i$) and $\tau\tau' \in \mathrm{prf}(\sigma)$ and $\sigma, \tau\tau' \models \beta$. Further, for every $\tau'' \in \mathrm{prf}(\tau')$, if $\varepsilon \preceq \tau''_i \prec \tau'_i$ then $\sigma, \tau\tau'' \models \alpha$.

We will say that a formula $\alpha \in \mathrm{DLTL}^{\otimes}(\widetilde{\Sigma})$ is *satisfiable* if there exist $\sigma \in \Sigma^\infty$ and $\tau \in \mathrm{prf}(\sigma)$ such that $\sigma, \tau \models \alpha$. The language defined by α is given by

$$L_\alpha \stackrel{\mathrm{def}}{=} \{\sigma \in \Sigma^\infty \mid \sigma, \varepsilon \models \alpha\}.$$

We say that $L \subseteq \Sigma^\infty$ is $\mathrm{DLTL}^{\otimes}(\widetilde{\Sigma})$-definable if there exists some $\alpha \in \mathrm{DLTL}^{\otimes}(\widetilde{\Sigma})$ with $L_\alpha = L$.

5 A Decision Procedure for $\mathrm{DLTL}^{\otimes}$

We will show the satisfiability problem for $\mathrm{DLTL}(\Sigma)$ is solvable in deterministic exponential time. This will be achieved by effectively constructing a product automaton $\mathcal{A}_\alpha$ for each $\alpha \in \mathrm{DLTL}^{\otimes}(\widetilde{\Sigma})$ such that the language accepted by $\mathcal{A}_\alpha$ is non-empty iff α is satisfiable. Our construction is a common generalization of the one for product LTL in [16] and the one for $\mathrm{DLTL}(\Sigma)$ in [5]. The solution to the satisfiability problem will at once lead to a solution to the model checking problem for programs modelled as synchronizing sequential agents.

Through the rest of the section we fix a formula $\alpha_0 \in \mathrm{DLTL}^{\otimes}$. In order to construct $\mathcal{A}_{\alpha_0}$ we first define the (Fischer-Ladner) closure of α_0. As a first step let $cl(\alpha_0)$ be the least set of formulas satisfying:

- $\alpha_0 \in cl(\alpha_0)$.
- $\sim\alpha \in cl(\alpha_0)$ implies $\alpha \in cl(\alpha_0)$.
- $\alpha \vee \beta \in cl(\alpha_0)$ implies $\alpha, \beta \in cl(\alpha_0)$.
- $\alpha\, \mathcal{U}_i^\pi \beta \in cl(\alpha_0)$ implies $\alpha, \beta \in cl(\alpha_0)$.

We will now take the *closure* of α_0 to be $CL(\alpha_0) = cl(\alpha_0) \cup \{\sim\alpha \mid \alpha \in cl(\alpha_0)\}$. From now on we shall identify $\sim\sim\alpha$ with α. Set $CL_i(\alpha_0) = CL(\alpha_0) \cap \mathrm{DLTL}_i^{\otimes}(\widetilde{\Sigma})$ for each i. We will often write CL instead of $CL(\alpha_0)$ and CL_i instead of $CL_i(\alpha_0)$. All formulas considered from now on will be assumed to belong to CL_i unless otherwise stated.

An *i-type atom* is a subset $A \subseteq CL_i$ which satisfies:

- $\top \in A$.

- $\alpha \in A$ iff $\sim\alpha \notin A$.
- $\alpha \vee \beta \in A$ iff $\alpha \in A$ or $\beta \in A$.
- $\beta \in A$ and $\varepsilon \in ||\pi||$ implies $\alpha\, \mathcal{U}_i^\pi \beta \in A$.

The set of i-type atoms is denoted AT_i. We next define the predicate Member$(\alpha, (A_1, \ldots, A_K))$ for each $\alpha \in CL(\alpha_0)$ and $(A_1, \ldots, A_K) \in AT_1 \times \ldots \times AT_K$. For convenience this predicate will be denoted as $\alpha \in (A_1, \ldots, A_K)$ and is given inductively by:

- Let $\alpha \in CL_i$. Then $\alpha \in (A_1, \ldots, A_K)$ iff $\alpha \in A_i$.
- Let $\alpha = \sim\beta$. Then $\alpha \in (A_1, \ldots, A_K)$ iff $\beta \notin (A_1, \ldots, A_K)$.
- Let $\alpha = \beta \vee \gamma$. Then $\alpha \in (A_1, \ldots, A_K)$ iff $\beta \in (A_1, \ldots, A_K)$ or $\gamma \in (A_1, \ldots, A_K)$.

The set of *i-type until requirements* is the subset of CL_i given by

$$Req_i = \{\alpha\, \mathcal{U}_i^\pi \beta \mid \alpha\, \mathcal{U}_i^\pi \beta \in CL_i\}.$$

We shall let ξ, ξ' range over Req_i. For each $\xi = \alpha\, \mathcal{U}_i^\pi \beta \in Req_i$ we fix a finite state automaton $\mathcal{A}_\xi$ such that $\mathcal{L}(\mathcal{A}_\xi) = ||\pi||$ where $\mathcal{L}(\mathcal{A}_\xi)$ is the language of finite words accepted by $\mathcal{A}_\xi$. We shall assume each such $\mathcal{A}_\xi$ is of the form $\mathcal{A}_\xi = (Q_\xi, \longrightarrow_\xi, I_\xi, F_\xi)$ where Q_ξ is the set of states, $\longrightarrow_\xi \subseteq Q_\xi \times \Sigma \times Q_\xi$ is the transition relation, $I_\xi \subseteq Q_\xi$ is the set of initial states and $F_\xi \subseteq Q_\xi$ is the set of final states. Without loss of generality, we shall assume that $\xi \neq \xi'$ implies $Q_\xi \cap Q_{\xi'} = \emptyset$ for every $\xi, \xi' \in Req_i$. We set $Q_i = \bigcup_{\xi \in Req_i} Q_\xi$ and $\widehat{Q}_i = Q_i \times \{0, 1\}$.

The product automaton $\mathcal{A}_{\alpha_0}$ associated with α_0 is now defined to be $\mathcal{A}_{\alpha_0} = (\{\mathcal{A}_i\}_{i \in Loc}, Q_{in})$ where for each i, $\mathcal{A}_i = (S_i, \Longrightarrow_i, F_i, F_i^\omega)$ is specified as

1. $S_i \subseteq AT_i \times 2^{Q_i} \times 2^{\widehat{Q}_i} \times \{\mathsf{stop}, \mathsf{go}\} \times \{0, 1\} \times \{\downarrow, \checkmark\}$ such that

   $$(A, X, \widehat{X}, s, x, f) \in S_i$$

   iff the following conditions are satisfied for each $\xi = \alpha\, \mathcal{U}_i^\pi \beta$:

   (i) If $\beta \in A$ then $F_\xi \subseteq X$. (Recall that $\mathcal{A}_\xi = (Q_\xi, \longrightarrow_\xi, I_\xi, F_\xi)$).

   (ii) If $\alpha \in A$ and $q \in X$ for some $q \in I_\xi$ then $\alpha\, \mathcal{U}_i^\pi \beta \in A$.

(iii) If $\alpha\,\mathcal{U}_i^{\pi}\beta \in A$ then either $\beta \in A$ and $\varepsilon \in ||\pi||$ or $(q, 1-x) \in \widehat{X}$ for some $q \in I_\xi$. (Note that we are considering the candidate $(A, X, \widehat{X}, s, x, f)$ for membership in S_i).

(iv) If $(q, z) \in \widehat{X}$ with $q \notin F_\xi$ or $\beta \notin A$ then $\alpha \in A$.

2. The transition relation $\Longrightarrow_i \subseteq S_i \times \Sigma_i \times S_i$ is defined as follows:

$$(A, X, \widehat{X}, s, x, f) \overset{a}{\Longrightarrow}_i (B, Y, \widehat{Y}, t, y, g)$$

iff the following conditions are satisfied for each $\xi = \alpha\,\mathcal{U}_i^{\pi}\beta \in Req_i$:

(i) $s = \mathsf{go}$.

(ii) Suppose $q' \in Q_\xi \cap Y$ and $q \overset{a}{\longrightarrow}_\xi q'$ and $\alpha \in A$. Then $q \in X$.

(iii) Suppose $(q, z) \in \widehat{X}$ with $q \in Q_\xi$. Suppose further that $q \notin F_\xi$ or $\beta \notin A$. Then $(q', z) \in \widehat{Y}$ for some q' with $q \overset{a}{\longrightarrow}_\xi q'$.

(iv) If $f = \checkmark$ then $(y, g) = (1 - x, \downarrow)$. If $f = \downarrow$ then,

$$(y, g) = \begin{cases} (x, \downarrow), & \text{if there exists } (q, x) \in \widehat{X} \text{ such that} \\ & q \notin F_\xi \text{ or } \beta \notin A \\ (x, \checkmark), & \text{otherwise.} \end{cases}$$

3. $F_i = \{(A, X, \widehat{X}, s, x, f) \mid s = \mathsf{stop} \text{ and } \widehat{X} = \emptyset\}$.

4. $F_i^{\omega} = \{(A, X, \widehat{X}, s, x, f) \mid f = \checkmark\}$.

Finally, $Q_{in} \subseteq Q_1 \times \ldots \times Q_K$ is specified as

$$((A_1, X_1, \widehat{X}_1, s_1, x_1, f_1), \ldots, (A_K, X_K, \widehat{X}_K, s_K, x_K, f_K)) \in Q_{in}$$

iff $\alpha_0 \in (A_1, \ldots, A_K)$ and $(x_i, f_i) = (0, \checkmark)$ for every $i \in Loc$.

The main result of this section can now be formulated.

Theorem 5.1 *$\mathcal{L}(\mathcal{A}_{\alpha_0}) = L_{\alpha_0}$ where $\mathcal{A}_{\alpha_0}$ is as defined above. Hence α_0 is satisfiable iff $\mathcal{L}(\mathcal{A}_{\alpha_0}) \neq \emptyset$. Moreover, the size of $\mathcal{A}_{\alpha_0}$ is $2^{O(|\alpha_0|)}$ and consequently the satisfiability problem for* DLTL$^{\otimes}(\widetilde{\Sigma})$ *is decidable in exponential time.*

Proof: Let $\sigma \in \mathcal{L}(\mathcal{A}_{\alpha_0})$ by the accepting run $\rho : \mathrm{prf}(\sigma) \to Q_G^{\mathcal{A}}$. For each $\tau \in \mathrm{prf}(\sigma)$ let $\rho(\tau)[i] = (A_{\tau_i}, X_{\tau_i}, \widehat{X}_{\tau_i}, s_{\tau_i}, x_{\tau_i}, f_{\tau_i})$. Then a detailed examination of the above construction reveals that for all $\tau \in \mathrm{prf}(\sigma)$ and $\delta \in CL_i$,

$$\sigma, \tau \models \delta \text{ iff } \delta \in A_{\tau_i}.$$

By definition of Q_{in} we are assured that $\alpha_0 \in (\rho(\varepsilon)[1], \ldots, \rho(\varepsilon)[K])$. Hence a simple induction on the structure of α_0 will show that $\sigma, \varepsilon \models \alpha_0$.

Conversely, if α_0 is satisfiable we may assume that $\sigma, \varepsilon \models \alpha_0$ for some σ. We will construct an accepting run $\rho : \mathrm{prf}(\sigma) \to Q_G^{\mathcal{A}}$. For each $\tau \in \mathrm{prf}(\sigma)$ and each i, we set $\rho(\tau)[i] = (A_{\tau_i}, X_{\tau_i}, \widehat{X}_{\tau_i}, s_{\tau_i}, x_{\tau_i}, f_{\tau_i})$ and define the various components of this tuple as follows. First we define A_{τ_i} by $A_{\tau_i} = \{\alpha \in CL_i \mid \sigma, \tau \models \alpha\}$. Next s_{τ_i} is defined as $s_{\tau_i} = \mathsf{stop}$ iff $\sigma \restriction i = \tau_i$ (recall the convention $\tau \restriction i = \tau_i$). In defining the other components it will be convenient to adopt the following terminology.

Let $\xi = \alpha\, \mathcal{U}_i^{\pi} \beta$ and $q \in Q_\xi$ and $\tau_i \in \Sigma_i^*$. Then an *accepting run of* $\mathcal{A}_\xi$ *over* τ_i *starting from* q is a map $R : \mathrm{prf}(\tau_i) \longrightarrow Q_\xi$ such that $R(\varepsilon) = q$, $R(\tau_i) \in F_\xi$ and $R(\tau_i'') \xrightarrow{a}_\xi R(\tau_i'' a)$ for every $\tau_i'' a \in \mathrm{prf}(\tau_i)$. In case $q \in I_\xi$ we shall just say that R is an *accepting run of* $\mathcal{A}_\xi$ *over* τ_i.

Let $\xi = \alpha\, \mathcal{U}_i^{\pi} \beta$ and $q \in Q_\xi$. Then $q \in X_{\tau_i}$ iff there exist τ' and R' such that $\tau\tau' \in \mathrm{prf}(\sigma)$, $\sigma, \tau\tau' \models \beta$, and for every $\tau'' \in \mathrm{prf}(\tau')$, if $\varepsilon \preceq \tau_i'' \prec \tau_i'$ then $\sigma, \tau\tau'' \models \alpha$. Furthermore, R' should be an accepting run of $\mathcal{A}_\xi$ over τ_i' starting from q.

To specify the remaining three components we shall make use of a chronicle of obligations.

We'll say that (τ, ξ) is an *obligation* if $\tau \in \mathrm{prf}(\sigma)$ and $\xi = \alpha\, \mathcal{U}_i^{\pi} \beta \in Req_i$ such that $\sigma, \tau \models \alpha\, \mathcal{U}_i^{\pi} \beta$ but $\sigma, \tau \not\models \beta$ or $\varepsilon \notin ||\pi||$. Let (τ, ξ) be an obligation. We shall say that the pair (τ', R') is a *witness* for (τ, ξ) iff $\tau\tau' \in \mathrm{prf}(\sigma)$ and $\sigma, \tau\tau' \models \beta$ and for every $\tau'' \in \mathrm{prf}(\tau')$, if $\varepsilon \preceq \tau_i'' \prec \tau_i'$ then $\sigma, \tau\tau'' \models \alpha$. Furthermore, $\tau_i' \in ||\pi||$ and R' is an accepting run of $\mathcal{A}_\xi$ over τ_i'. A *chronicle set* CH is a set of quadruples satisfying that if $(\tau, \xi, \tau', R') \in CH$ then (τ, ξ) is an obligation and (τ', R') is witness for (τ, ξ). Moreover, for every obligation (τ, ξ) there is a unique element of the form (τ, ξ, τ', R') in CH. We fix such a set CH which clearly exists.

Now x_{τ_i} and f_{τ_i} are defined by mutual induction as follows. For the base case, $(x_\varepsilon, f_\varepsilon) = (0, \checkmark)$. For the induction step, let $\tau = \tau' a$. Suppose $a \notin \Sigma_i$. Then $(x_{\tau_i}, f_{\tau_i}) = (x_{\tau_i'}, f_{\tau_i'})$. So assume that $a \in \Sigma_i$. If $f_{\tau_i'} = \checkmark$ then $(x_{\tau_i}, f_{\tau_i}) = (1 - x_{\tau_i'}, \downarrow)$. Suppose $f_{\tau_i'} = \downarrow$. Then $(x_{\tau_i}, f_{\tau_i}) = (x_{\tau_i'}, \downarrow)$ if there exists $(\tau'', \xi_1, \tau''', R_1') \in CH$ such that $\tau'' \preceq \tau' \prec \tau''\tau'''$ and $x_{\tau_i''} = 1 - x_{\tau_i'}$. Otherwise, $f_{\tau_i} = \checkmark$ and $x_{\tau_i} = x_{\tau_i'}$.

The only remaining component to be dealt with is $\widehat{X}_{\tau_i}$. This is now defined via: $(q, z) \in \widehat{X}_{\tau_i}$ iff there exists $(\tau', \xi, \tau'', R_1') \in CH$ such that for some $\tau''' \in \mathrm{prf}(\tau'')$, $\tau_i' \preceq \tau_i = \tau_i'\tau_i'''$ and furthermore $R_1'(\tau_i''') = q$ and $x_{\tau_i'} = 1 - z$. Using these definitions it is not difficult to show that ρ is an accepting run.

Finally, by Lemma 3.5 it suffices to show that our construction yields a product automaton of exponential size. Clearly, $CL(\alpha_0)$ is linear in α_0, and surely then $|AT_1| + \ldots + |AT_K| = 2^{O(|\alpha_0|)}$. Moreover, it is well-known that each $\pi \in \mathrm{Prg}(\Sigma_i)$ in polynomial time can be converted to a finite (non-deterministic) automaton with a linear state space (see [6] for a recent account of such conversions). Then both $Q_1 + \ldots + Q_K$ and $\widehat{Q}_1 + \ldots + \widehat{Q}_K$ are of size $O(|\alpha_0|)$. Consequently, $|Q_G^{\mathcal{A}}| = 2^{O(|\alpha_0|)}$ as required. $\square$

The procedure outlined above also lends itself to a solution to the *model checking problem*, which is defined as for DLTL except that a finite-state program is now simply a product automaton $\mathcal{P}$. Once again, we do not wish to enter into details.

6 An Expressiveness Result

We now wish to show that our logic is expressively complete with respect to the regular product languages. In fact we will identify a natural sublogic — to be denoted $\mathrm{DLTL}_{-}^{\otimes}$ — which also enjoys this property.

The syntax of the formulas of $\mathrm{DLTL}_{-}^{\otimes}(\widetilde{\Sigma})$ remains as for $\mathrm{DLTL}^{\otimes}(\widetilde{\Sigma})$, but the until modality is to be restricted to the derived operator $\langle\_\rangle_i$. Formally, the set of formulas and locations of this sublogic is obtained via:

- $\top$ is a formula and $\mathrm{loc}(\top) = \emptyset$.
- Suppose α and β are formulas so are $\sim\alpha$ and $\alpha \vee \beta$. Moreover $\mathrm{loc}(\sim\alpha) = \mathrm{loc}(\alpha)$ and $\mathrm{loc}(\alpha \vee \beta) = \mathrm{loc}(\alpha) \cup \mathrm{loc}(\beta)$.
- Suppose α is formula such that $\mathrm{loc}(\alpha) \subseteq \{i\}$ and $\pi \in \mathrm{Prg}(\Sigma_i)$. Then $\langle\pi\rangle_i\alpha$ is formula. Moreover, $\mathrm{loc}(\langle\pi\rangle_i\alpha) = \{i\}$.

Proposition 6.1 *If $L \in \mathcal{R}^{\otimes}(\widetilde{\Sigma})$ then L is $\mathrm{DLTL}_{-}^{\otimes}(\widetilde{\Sigma})$-definable.*

Proof: It suffices to show that the claim holds for $L \in \mathcal{R}_0^{\otimes}(\widetilde{\Sigma})$ because each member of $\mathcal{R}^{\otimes}(\widetilde{\Sigma})$ is a finite union of languages in $\mathcal{R}_0^{\otimes}(\widetilde{\Sigma})$.

Let $L = L_1 \otimes \ldots \otimes L_K \in \mathcal{R}_0^{\otimes}(\widetilde{\Sigma})$. Then each $L_i \cap \Sigma_i^*$ is regular. Clearly $L_i \cap \Sigma_i^* = ||\pi_i||$ for some $\pi_i \in \text{Prg}(\Sigma_i)$. Now define $\alpha_*^i = \langle \pi_i \rangle_i [\pi_i']_i \bot$ where $\pi_i' = (a_1 + \ldots + a_n)$ with $\Sigma_i = \{a_1, \ldots, a_n\}$.

Next, $L_i \cap \Sigma_i^\omega$ is ω-regular. Hence it is accepted, due to McNaughton's theorem [10], by a *deterministic* Muller automaton. Choose such an automaton $\mathcal{M} = (Q, q_{in}, \longrightarrow, \mathcal{F})$, which we, without loss of generality, assume to be complete. (See [5] for the formal details). For $q, q' \in Q$ we set $L_{q,q'} = \{\tau \mid q \stackrel{\tau}{\longrightarrow} q'\}$, which is obviously a regular subset of Σ_i^*. So we can fix $\pi_{q,q'} \in \text{Prg}(\Sigma_i)$ such that $L_{q,q'} = ||\pi_{q,q'}||$. Moreover, by the determinacy of $\mathcal{M}$ it follows that $L_{q,q'} \cap L_{q,q''} \neq \emptyset$ implies $q' = q''$. We now define

$$\alpha_\omega^i = \bigvee_{F \in \mathcal{F}} \bigvee_{q \in F} \langle \pi_{q_{in},q} \rangle_i \left(\bigwedge_{q' \notin F} [\pi_{q,q'}]_i \bot \wedge \bigwedge_{j=0}^{n-1} [\pi_{q,q_j}]_i \langle \pi_{q_j, q_{j \oplus 1}} \rangle_i \top \right)$$

with the assumption $\{q_0, q_1, \ldots, q_{n-1}\}$ is an enumeration of the $F \in \mathcal{F}$ under consideration and "$\oplus$" denotes addition modulo n. It is easy to show that $\sigma \restriction i \in L_i \cap \Sigma_i^\omega$ iff $\sigma, \varepsilon \models \alpha_\omega^i$.

The required formula α is given by $\alpha = \bigwedge_{i \in Loc} \alpha^i$ where $\alpha^i = \alpha_*^i \vee \alpha_\omega^i$ for each i. It is a routine exercise to establish $L_\alpha = L_1 \otimes \ldots \otimes L_K$.

$\Box$

On the other hand, Theorem 3.4 together with Theorem 5.1 states that L_{α_0} is a product language over $\widetilde{\Sigma}$ for any $\alpha_0 \in \text{DLTL}^{\otimes}(\widetilde{\Sigma})$. Since $\text{DLTL}_-^{\otimes}$ is a sublogic of $\text{DLTL}^{\otimes}$ we have the following expressiveness result.

Corollary 6.2 *Let $L \subseteq \Sigma^\infty$. Then the following statements are equivalent:*

(i) $L \in \mathcal{R}^{\otimes}(\widetilde{\Sigma})$.

(ii) L *is* $\text{DLTL}_-^{\otimes}(\widetilde{\Sigma})$*-definable.*

(iii) L *is* $\text{DLTL}^{\otimes}(\widetilde{\Sigma})$*-definable.*

7 Discussion

We shall conclude this section by placing regular product languages in the broader context of regular Mazurkiewicz trace languages. For an introduction to (Mazurkiewicz) traces related to the concerns of the present paper, we refer the reader to [11]. We shall assume the bare minimum of the background material on traces.

We begin by noting that the distributed alphabet $\widetilde{\Sigma} = \{\Sigma_i\}_{i=1}^{K}$ induces the trace alphabet $(\Sigma, I_{\widetilde{\Sigma}})$ where the irreflexive and symmetric independence relation $I_{\widetilde{\Sigma}} \subseteq \Sigma \times \Sigma$ is given by:

$$a \; I_{\widetilde{\Sigma}} \; b \;\text{ iff }\; Loc(a) \cap Loc(b) = \emptyset.$$

Recall that $Loc(x) = \{i \mid x \in \Sigma_i\}$ for $x \in \Sigma$. We shall write I instead of $I_{\widetilde{\Sigma}}$ from now on. This independence relation in turn induces the equivalence relation $\approx_I \subseteq \Sigma^\infty \times \Sigma^\infty$ (from now on written as $\approx$) given by:

$$\sigma \approx \sigma' \;\text{ iff }\; \sigma \restriction i = \sigma' \restriction i \;\text{ for every }\; i \in Loc.$$

The $\approx$-equivalence classes of Σ^∞ constitute the set of finite and infinite traces generated by the trace alphabet (Σ, I). Traces can be — upto isomorphisms — uniquely represented as certain Σ-labelled posets where the labelling functions respect I in a natural manner. A trace language is just a subset of $\Sigma^\infty / \approx$.

A language $L \subseteq \Sigma^\infty$ is trace consistent in case $\sigma \in L$ and $\sigma \approx \sigma'$ implies $\sigma' \in L$, for every σ, σ'. The point is, a trace consistent language L canonically represents the trace language $\{[\sigma]_\approx \mid \sigma \in L\}$. We extend this idea to logical formulas by saying that $\alpha \in \mathrm{DLTL}^{\otimes}(\widetilde{\Sigma})$ is trace consistent iff L_α is trace consistent. It is easy to show that *every* formula of $\mathrm{DLTL}^{\otimes}(\widetilde{\Sigma})$ is trace consistent. An important feature of properties defined by trace consistent formulas is that they can often be verified efficiently using partial order based reduction techniques [3, 12, 18]. Consequently, $\mathrm{DLTL}^{\otimes}(\widetilde{\Sigma})$ provides a flexible and powerful means for specifying trace consistent properties of distributed programs. As it turns out, every formula of $\mathrm{DLTL}^{\otimes}(\widetilde{\Sigma})$ defines — via the canonical representation — a regular trace language contained in $\Sigma^\infty / \approx$. Hence by Corollary 6.2, every regular product language corresponds to a regular trace language.

The converse however is not true. To bring this out, consider the distributed alphabet $\widetilde{\Sigma} = \{\{a, a', c\}, \{b, b', c\}\}$ and the language $L = \{cab, cba, ca'b', cb'a'\}^\omega$. Then it is easy to check that L is trace consistent and ω-regular and that it is *not* a regular product language. In a forthcoming paper we shall deal with the problem of extending $\mathrm{DLTL}^{\otimes}(\widetilde{\Sigma})$ so as to capture *all* of the regular trace languages.

References

[1] Fischer, M. J., Ladner, R. E.: Propositional dynamic logic of regular programs. Journal of Computer and System Sciences **18**(2) (1979) 194–211

[2] Gabbay, A., Pnueli, A., Shelah, S., Stavi, J.: On the temporal analysis of fairness. Proceedings of the 7th Annual Symposium on Principles of Programming Languages, ACM (1980) 163–173

[3] Godefroid, P.: Partial-order Methods for the Verification of Concurrent Systems. Lecture Notes in Computer Science 1032, Springer-Verlag (1996)

[4] Harel, D.: Dynamic logic. In Gabbay, D., Guenthner, F., eds.: Handbook of Philosophical Logic, Vol. II, Reidel, Dordrecht (1984) 497–604

[5] Henriksen, J. G., Thiagarajan, P. S.: Dynamic linear time temporal logic. BRICS technical report RS-97-8, Department of Computer Science, University of Aarhus, Denmark (1997)

[6] Hromkovič, J., Seibert, S., Wilke, T.: Translating regular expressions into small ε-free nondeterministic automata. Proceedings of the 12th Annual Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science 1200, Springer-Verlag (1997) 55–66

[7] Kamp, H. R.: Tense Logic and the Theory of Linear Order. Ph.D. thesis, University of California (1968)

[8] Manna, Z., Pnueli, A.: The Temporal Logic of Reactive and Concurrent Systems (Specification), Springer-Verlag (1992)

[9] Mazurkiewicz, A.: Concurrent program schemes and their interpretations. Technical report DAIMI PB-78, Department of Computer Science, University of Aarhus, Denmark (1977)

[10] McNaughton, R.: Testing and generating infinite sequences by a finite automaton. Information and Control **9** (1966) 521–530

[11] Mukund, M., Thiagarajan, P. S.: Linear time temporal logics over Mazurkiewicz traces. Proceedings of the 21st Intl. Symposium on

Mathematical Foundations of Computer Science, Lecture Notes in Computer Science 1113, Springer-Verlag (1996) pp. 62–92

[12] Peled, D.: Partial order reduction: model checking using representatives. Proceedings of the 21st Intl. Symposium on Mathematical Foundations of Computer Science, Lecture Notes in Computer Science 1113, Springer-Verlag (1996) 93–112

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

[14] Thiagarajan, P. S.: A trace based extension of linear time temporal logic. Proceedings of the 9th Annual Symposium on Logic in Computer Science, IEEE (1994) 438–447

[15] Thiagarajan, P. S.: PTL over product state spaces. Technical report TCS-95-4, School of Mathematics, SPIC Science Foundation, Madras (1995)

[16] Thiagarajan, P. S.: A trace consistent subset of PTL. Proceedings of the 6th International Conference on Concurrency Theory, Lecture Notes in Computer Science 962, Springer-Verlag (1995) 438–452

[17] Thomas, W.: Automata over infinite objects. In van Leeuwen, J., ed., Handbook of Theoretical Computer Science, Vol. B: Formal Models and Semantics, Elsevier/MIT Press (1990) 133–191

[18] Valmari, A.: A stubborn attack on state explosion. Formal Methods in Systems Design **1** (1992) 285–313

[19] Vardi, M. Y., Wolper, P.: An automata-theoretic approach to automatic program verification. Proceedings of the 1st Annual Symposium on Logic in Computer Science, IEEE (1986) 332–345

[20] Wolper, P.: Temporal logic can be more expressive. Proceedings of the 22nd Annual Symposium on Foundations of Computer Science, IEEE (1981) 340–348

[21] Wolper, P., Vardi, M. Y., Sistla, A. P.: Reasoning about infinite computation paths. Proceedings of the 24nd Annual Symposium on Foundations of Computer Science, IEEE (1983) 185–194

Recent BRICS Report Series Publications

RS-97-9 **Jesper G. Henriksen and P. S. Thiagarajan.** ***A Product Version of Dynamic Linear Time Temporal Logic*****. April 1997. 18 pp. To appear in** ***Concurrency Theory: 7th International Conference*****, CONCUR '97 Proceedings, LNCS, 1997.**

RS-97-8 **Jesper G. Henriksen and P. S. Thiagarajan.** ***Dynamic Linear Time Temporal Logic*****. April 1997. 33 pp.**

RS-97-7 **John Hatcliff and Olivier Danvy.** ***Thunks and the λ-Calculus (Extended Version)*****. March 1997. 55 pp. Extended version of article to appear in the** ***Journal of Functional Programming*****.**

RS-97-6 **Olivier Danvy and Ulrik P. Schultz.** ***Lambda-Dropping: Transforming Recursive Equations into Programs with Block Structure*****. March 1997. 53 pp. Extended version of an article to appear in the 1997 ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation (PEPM '97), Amsterdam, The Netherlands, June 1997.**

RS-97-5 **Kousha Etessami, Moshe Y. Vardi, and Thomas Wilke.** ***First-Order Logic with Two Variables and Unary Temporal Logic*****. March 1997. 18 pp. To appear in** ***Twelfth Annual IEEE Symposium on Logic in Computer Science*****, LICS '97 Proceedings.**

RS-97-4 **Richard Blute, Josée Desharnais, Abbas Edalat, and Prakash Panangaden.** ***Bisimulation for Labelled Markov Processes*****. March 1997. 48 pp. To appear in** ***Twelfth Annual IEEE Symposium on Logic in Computer Science*****, LICS '97 Proceedings.**

RS-97-3 **Carsten Butz and Ieke Moerdijk.** ***A Definability Theorem for First Order Logic*****. March 1997. 10 pp.**

RS-97-2 **David A. Schmidt.** ***Abstract Interpretation in the Operational Semantics Hierarchy*****. March 1997. 33 pp.**

RS-97-1 **Olivier Danvy and Mayer Goldberg.** ***Partial Evaluation of the Euclidian Algorithm (Extended Version)*****. January 1997. 16 pp. To appear in the journal** ***Lisp and Symbolic Computation*****.**

RS-96-62 **P. S. Thiagarajan and Igor Walukiewicz.** ***An Expressively Complete Linear Time Temporal Logic for Mazurkiewicz Traces*****. December 1996. i+13 pp. To appear in** ***Twelfth Annual IEEE Symposium on Logic in Computer Science*****, LICS '97 Proceedings.**