Basic Research in Computer Science

BRICS RS-98-8 Thiagarajan & Henriksen: Distributed Versions of Linear Time Temporal Logic

Distributed Versions of Linear Time Temporal Logic: A Trace Perspective

P. S. Thiagarajan
Jesper G. Henriksen

BRICS Report Series **RS-98-8**

ISSN 0909-0878 **April 1998**

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/98/8/`

Distributed Versions of Linear Time Temporal Logic: A Trace Perspective*

P. S. Thiagarajan†
SPIC Mathematical Institute, Chennai, India
pst@smi.ernet.in

Jesper Gulmann Henriksen
BRICS‡, Department of Computer Science,
University of Aarhus, Denmark
gulmann@brics.dk

April 28, 1998

1 Introduction

Linear time Temporal Logic (LTL) as proposed by Pnueli [37] has become a well established tool for specifying the dynamic behaviour of distributed systems. A basic feature of LTL is that its formulas are interpreted over sequences. Typically, such a sequence will model a computation of a system; a sequence of states visited by the system or a sequence of actions executed by the system during the course of the computation. A system is said to satisfy a specification expressed as an LTL formula in case every computation of the system is a model of the formula. A rich theory of LTL is now available using which one can effectively verify whether a finite state system meets its specification [51]. Indeed, the verfication task can be automated (for instance using the software packages SPIN [21] and FormalCheck [2]) to handle large systems of practical interest.

*Appears as a chapter of "Lectures on Petri Nets I: Basic Models", Lecture Notes in Computer Science 1491, Springer-Verlag (1998), pp. 643-681.
†This work has been supported by BRICS and IFCPAR Project 1502-1.
‡**B**asic **R**esearch **i**n **C**omputer **S**cience, Centre of the Danish National Research Foundation.

In many applications the computations of a distributed system will constitute interleavings of the occurrences of causally independent actions. Consequently, the computations can be naturally grouped together into equivalence classes where two computations are equated in case they are two different interleavings of the same partially ordered stretch of behaviour. It turns out that many of the properties expressed as LTL-formulas happen to have the so called "all-or-none" property. Either all members of an equivalence class of computations will have the desired property or none will do ("leads to deadlock" is one such property). For verifying such properties one has to check the property for just one member of each equivalence class. This is the insight underlying many of the partial-order based verification methods [17, 35, 50]. As may be guessed, the importance of these methods lies in the fact that via these methods the computational resources required for the verification task can often be dramatically reduced.

It is often the case that the equivalence classes of computations generated by a distributed system constitute objects called Mazurkiewicz traces. They can be canonically represented as restricted labelled partial orders. This opens up an alternative way of exploiting the non-sequential nature of the computations of a distributed systems and the attendant partial-order based methods. It consists of developing linear time temporal logics that can be directly interpreted over Mazurkiewicz traces. In these logics, every specification is guaranteed to have the "all-or-none" property and hence can take advantage of the partial-order based reduction methods during the verification process. The study of these logics also exposes the richness of the partial-order settings from a logical standpoint and the complications that can arise as a consequence.

Our aim here is to present an overview of linear time temporal logics whose models can be viewed as Mazurkiewicz traces. The presentation is, in principle, self-contained though previous exposure to temporal logics [12] and automata over infinite objects [49] will be very helpful. We have provided net-theoretic examples whenever possible in order to emphasize the broad scope of applicability of the material.

In the next section we introduce linear time temporal logic and sketch the automata-theoretic solutions to the satisfiability problem (does a formula have a model?) and the model checking problem (do all computations of a system constitute models of a given specification formula?). In Section 3 we introduce Mazurkiewicz traces viewed as equivalence classes of sequences. This leads to the precise formualtion of the notion "all-or-none" LTL properties.

Next we introduce a well-understood class of trace languages called product languages. The automata that recognize these languages are called prod-

uct automata and they incorporate a simple and yet useful method of forming distributed systems. The system consists of a network of sequential agents, each with its own alphabet of actions. In the interesting instances the alphabets are not pair-wise disjoint. One then imposes a synchronization regime under which the agents are forced to carry out common actions together. After presenting a theory of product languages and automata, we formulate in Section 5 a simple version of a trace-based version of LTL called product LTL. The formulas of this logic have a natural semantics in terms of the computations generated by a network of sequential agents as introduced in the previous section. Using the theory of product automata we then provide solutions to the satisfiability and model checking problems for product LTL.

In Section 6 we introduce the representation of Mazurkiewicz traces as restricted labelled partial orders. We then provide a rapid introduction to the theory of trace languages and automata that we call asynchronous automata for recognizing trace languages. In the subsequent section we introduce the logic TrPTL which is a trace-based logic with much richer possibilities than product LTL. We then provide solutions to the satisfiability and model checking problems for TrPTL using asynchronous automata. This is followed by a brief survey of other trace-based linear time temporal logics available in the literature. Section 8 is devoted to considering various expressiveness issues associated with our temporal logics. We conclude in the final section with remarks about branching time temporal logics based on traces.

2 Linear Time Temporal Logic

In our formulation of linear time temporal logics it will be convenient to treat *actions* as first class objects both at the syntactic and semantic levels. As a first step we shall consider a version of LTL (linear time temporal logic) in which the next-state modality is indexed by actions.

Through the rest of the paper we fix a finite non-empty alphabet of actions 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 $\omega = \{0, 1, 2, \ldots\}$. We set $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$ and denote the null word by ε. We let σ, σ' range over Σ^ω and τ, τ', τ'' range over Σ^*. Finally $\preceq$ is the usual prefix ordering defined over Σ^* and for $u \in \Sigma^\infty$, we let $\mathrm{prf}(u)$ be the set of finite prefixes of u.

Next we fix a finite non-empty set of atomic propositions $P = \{p_1, p_2, \ldots\}$ and let p, q range over P. The set of formulas of LTL(Σ) is then given by the syntax:

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

Through the rest of this section α, β will range over LTL(Σ).

A model of LTL(Σ) is a pair $M = (\sigma, V)$ where $\sigma \in \Sigma^\omega$ and $V : \text{prf}(\sigma) \to 2^P$ is a valuation function. Let $M = (\sigma, V)$ be a model, $\tau \in \text{prf}(\sigma)$ and α be a formula. Then $M, \tau \models \alpha$ will stand for α being satisfied at τ in M. This notion is defined inductively in the expected manner.

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

Along with the usual propositional connectives $\wedge, \supset$ and $\equiv$ we will also use the propositional constants, $\top \stackrel{\Delta}{\iff} p_1 \vee \sim p_1$ and $\bot \stackrel{\Delta}{\iff} \sim \top$. Some useful derived modalities are:

- $O\alpha \stackrel{\Delta}{\iff} \bigvee_{a \in \Sigma} \langle a \rangle \alpha$.
- $\Diamond\alpha \stackrel{\Delta}{\iff} \top U \alpha$.
- $\Box\alpha \stackrel{\Delta}{\iff} \sim\Diamond \sim\alpha$.

Let $M = (\sigma, V)$ be a model and $\tau \in \text{prf}(\sigma)$. Then it is easy to check the following assertions.

- $M, \tau \models O\alpha$ iff $M, \tau' \models \alpha$ where $\tau' \in \text{prf}(\sigma)$ is such that $|\tau'| = |\tau| + 1$.
- $M, \tau \models \Diamond\alpha$ iff there exists a $\tau' \in \Sigma^*$ with $\tau\tau' \in \text{prf}(\sigma)$ such that $M, \tau\tau' \models \alpha$.
- $M, \tau \models \Box\alpha$ iff for each $\tau' \in \Sigma^*$, $\tau\tau' \in \text{prf}(\sigma)$ implies $M, \tau\tau' \models \alpha$.

Note that $O\alpha$ is the usual next-state operator of LTL.

We say that a formula $\alpha \in$ LTL(Σ) is *satisfiable* iff there exist a model $M = (\sigma, V)$ and $\tau \in \text{prf}(\sigma)$ such that $M, \tau \models \alpha$. This logic does not refer to the past either in the syntax or in the semantics. Hence the formula α is satisfiable iff there exists a model M such that $M, \varepsilon \models \alpha$. This is easy to check. The *satisfiability problem* for LTL is to develop a decision procedure

which will determine whether a given formula α is satisfiable. We will later in this section describe such a decision procedure.

We now wish to formulate the model checking problem for LTL(Σ). A *finite-state program* over Σ is a structure $Pr = (S, \longrightarrow, S_{in}, V_{Pr})$ where:

- S is a finite set of states.
- $\longrightarrow \subseteq S \times \Sigma \times S$ is a transition relation.
- $S_{in} \subseteq S$ is a set of initial states of the program.
- $V_{Pr} : S \to 2^P$ assigns a subset of P to each state of the program.

The members of P capture a finite set of basic assertions concerning the program which can usually be "read off" by examining the states of Pr and this is described by V_{Pr}. It will often be the case that the set of initial states is a singleton.

It is easy to arrange matters so that at each reachable state of the program at least one transition can be performed. We will assume that this is indeed the case for all program models we consider in this paper. Further we will say "program" instead "finite-state program" from now on.

A *computation* of the program Pr is a pair (σ, ρ) where $\sigma \in \Sigma^\omega$ and $\rho : \mathrm{prf}(\sigma) \to S$ is a map which satisfies:

- $\rho(\varepsilon) \in S_{in}$.
- $\rho(\tau) \stackrel{a}{\longrightarrow} \rho(\tau a)$ for each $\tau a \in \mathrm{prf}(\sigma)$.

Let (σ, ρ) be a computation of the program Pr. Then this computation canonically induces the model $M_{\sigma,\rho} = (\sigma, V_\rho)$ where V_ρ is given by: $V_\rho(\tau) = V_{Pr}(\rho(\tau))$ for each $\tau \in \mathrm{prf}(\sigma)$.

Let Pr be a program and α be a formula of LTL(Σ). We say that Pr *meets the specification* α — denoted $Pr \models \alpha$ — if for every computation (σ, ρ) of Pr, it is the case that $M, \varepsilon \models \alpha$ where M is the model induced by the computation (σ, ρ). The *model checking problem* is to decide for a given program Pr and a given formula α whether or not $Pr \models \alpha$. We will sketch a solution to the model checking problem later in this section.

Let $\mathcal{N} = (B, E, F, c_{in})$ be a finite elementary net system. In other words, it is an elementary net system in which both B, the set of conditions and E, the set of events are finite sets. We can associate the program $Pr_{\mathcal{N}} = (S, \longrightarrow, S_{in}, V_{Pr})$ with $\mathcal{N}$ as follows:

- $\Sigma = E$ and $P = B$.

- S is the least subset of 2^B and $\longrightarrow$ is the least subset of $S \times \Sigma \times S$ satisfying:
  - $c_{in} \in S$.
  - Suppose $c \in S$ and $e \in E$ such that ${}^\bullet e \subseteq c$ and $e^\bullet \cap c = \emptyset$. Then $c' \in S$ and $(c, e, c') \in \longrightarrow$ where $c' = (c - {}^\bullet e) \cup e^\bullet$.
- $S_{in} = \{c_{in}\}$.
- $V_{Pr}(c) = c$ for every $c \in S$.

Thus the so called case graph is the underlying transition system of the program. The conditions serve as the atomic propositions.

For $c \subseteq B$, let α_c be the formula $\bigwedge_{b \in c} b$. Now consider the specification $\Box \sim \alpha_c$ for some $c \subseteq B$. Then $Pr_{\mathcal{N}} \not\models \Box \sim \alpha_c$ iff c is a reachable state (i.e. $c \in S$) in $\mathcal{N}$. Next suppose e and e' are two events. Then $Pr_{\mathcal{N}} \models \Box\Diamond\langle e\rangle\top \supset \Box\Diamond\langle e'\rangle\top$ captures the fact that in $\mathcal{N}$, along every computation, if e occurs infinitely often then so does e'. A rich variety of liveness and safety properties can be expressed in LTL(Σ). For a substantial collection of examples the reader should see [26].

It turns out that both the satisfiability and model checking problems for LTL can be solved elegantly using Büchi automata [51]. We start with a brief introduction to these automata. A *Büchi automaton* over Σ is a tuple $\mathcal{B} = (Q, \longrightarrow, Q_{in}, F)$ where:

- Q is a finite non-empty set of states.
- $\longrightarrow \subseteq Q \times \Sigma \times Q$ is a transition relation.
- $Q_{in} \subseteq Q$ is a set of initial states.
- $F \subseteq Q$ is a set of accepting states.

Let $\sigma \in \Sigma^\omega$. Then a *run* of $\mathcal{B}$ over σ is a map $\rho : \mathrm{prf}(\sigma) \longrightarrow Q$ such that:

- $\rho(\varepsilon) \in Q_{in}$.
- $\rho(\tau) \xrightarrow{a} \rho(\tau a)$ for each $\tau a \in \mathrm{prf}(\sigma)$.

The run ρ is *accepting* iff $\inf(\rho) \cap F \neq \emptyset$ where $\inf(\rho) \subseteq Q$ is given by $q \in \inf(\rho)$ iff $\rho(\tau) = q$ for infinitely many $\tau \in \mathrm{prf}(\sigma)$. Finally $\mathcal{L}(\mathcal{B})$, the *language of ω-words accepted by $\mathcal{B}$*, is:

$$\mathcal{L}(\mathcal{B}) = \{\sigma \mid \exists \text{ an accepting run of } \mathcal{B} \text{ over } \sigma\}.$$

The languages recognized by Büchi automata are called the *ω-regular languages*. For an excellent survey of regular languages and automata over infinite objects, the reader is referred to [49].

It is easy to solve the *emptiness problem* for Büchi automata; to determine whether or not the language accepted by a Büchi automaton is empty. This can be done in time linear in the size of the automaton where the size of a Büchi automaton is the number of states of the automaton [49].

We will now show how one can effectively construct for each $\alpha \in \mathrm{LTL}(\Sigma)$, a Büchi automaton $\mathcal{B}_\alpha$ such that the language of ω-words accepted by $\mathcal{B}_\alpha$ is non-empty iff α is satisfiable. This is an action-based version of the elegant solution presented in [51] for LTL.

Through the rest of the section we fix a formula α_0. To construct $\mathcal{B}_{\alpha_0}$ we first define the (Fischer-Ladner) closure of α_0. For convenience we will assume that the derived next-state modality modality O is included in the syntax of $\mathrm{LTL}(\Sigma)$. We take $cl(\alpha_0)$ to be the least set of formulas that satisfies:

- $\alpha_0 \in cl(\alpha_0)$.
- If $\sim\beta \in cl(\alpha_0)$ then $\beta \in cl(\alpha_0)$.
- If $\alpha \vee \beta \in cl(\alpha_0)$ then $\alpha, \beta \in cl(\alpha_0)$.
- If $\langle a\rangle\alpha \in cl(\alpha_0)$ then $\alpha \in cl(\alpha_0)$.
- If $\alpha\ U\ \beta \in cl(\alpha_0)$ then $\alpha, \beta \in cl(\alpha_0)$. In addition, $O(\alpha\ U\ \beta) \in cl(\alpha_0)$.

Now $CL(\alpha_0)$, the *closure* of α_0, is defined to be:

$$CL(\alpha_0) = cl(\alpha_0) \cup \{\sim\beta \mid \beta \in cl(\alpha_0)\}.$$

In what follows $\sim\sim\beta$ will be identified with β. Moreover, throughout the section, all the formulas that we encounter will be assumed to be members of $CL(\alpha_0)$. For convenience, we shall often write CL instead of $CL(\alpha_0)$.

$A \subseteq CL$ is called an *atom* iff it satisfies :

- $\beta \in A$ iff $\sim\beta \notin A$.
- $\alpha \vee \beta \in A$ iff $\alpha \in A$ or $\beta \in A$.
- $\alpha\ U\ \beta \in A$ iff $\beta \in A$ or $\alpha, O(\alpha\ U\ \beta) \in A$.
- If $\langle a\rangle\alpha \in A$ and $\langle b\rangle\beta \in A$ then $a = b$.

$AT(\alpha_0)$ is the set of atoms and again we shall often write AT instead of $AT(\alpha_0)$. Finally we set U_{α_0}, the set of *until requirements* of α_0, to be the given by $U_{\alpha_0} = \{\alpha\ U\ \beta \mid \alpha\ U\ \beta \in CL\}$. We will often write U_0 instead of U_{α_0}.

The Büchi automaton $\mathcal{B}_{\alpha_0}$ (from now on denoted as $\mathcal{B}$) is now defined as $\mathcal{B} = (Q, \longrightarrow, Q_{in}, F)$, where the various components of $\mathcal{B}$ are specified as follows.

- $Q = AT \times 2^{U_0}$ is the set of states.
- The transition relation $\longrightarrow \subseteq Q \times \Sigma \times Q$ is given by $(A, x) \xrightarrow{a} (B, y)$ iff the following requirements are met:
  - For every $\langle a\rangle\alpha \in CL$, $\langle a\rangle\alpha \in A$ iff $\alpha \in B$ and for every $O(\alpha) \in CL$, $O(\alpha) \in A$ iff $\alpha \in B$.
  - if $\langle b\rangle\beta \in A$ then $b = a$.
  - if $x \neq \emptyset$ then $y = \{\alpha\ U\ \beta \mid \alpha\ U\ \beta \in x \text{ and } \beta \notin B\}$. If $x = \emptyset$ then $y = \{\alpha\ U\ \beta \mid \alpha\ U\ \beta \in B \text{ and } \beta \notin B\}$.
- $Q_{in} \subseteq Q$ is given by $(A, x) \in Q_{in}$ iff $\alpha_0 \in A$ and $x = \emptyset$.
- $F \subseteq Q$ is given by $(A, x) \in F$ iff $x = \emptyset$.

It is easy to show that $\mathcal{L}(\mathcal{B}) \neq \emptyset$ iff α_0 is satisfiable. It is also easy to check that the size of $\mathcal{B}$ is at most exponential in the size of α_0. As observed earlier the emptiness problem for a Büchi automaton can be solved in time linear in the size of the automaton. Thus we arrive at:

Theorem 2.1 *The satisfiability problem for* LTL(Σ) *is decidable in exponential time.*

Turning now to the model checking problem we first recall that the *intersection problem* for Büchi automata can be easily solved. In other words, let $\mathcal{B}_1, \mathcal{B}_2$ be two Büchi automata both operating over Σ. Then one can effectively construct a Büchi automaton $\mathcal{B}$ over the same alphabet such that the language accepted by $\mathcal{B}$ is the intersection of the languages accepted by $\mathcal{B}_1$ and $\mathcal{B}_2$. Moreover, the size of $\mathcal{B}$ can be assumed to be bounded by $2n_1n_2$ where n_1 is the size of $\mathcal{B}_1$ and n_2 is the size of $\mathcal{B}_2$ [49].

Now let $Pr = (S, \longrightarrow, S_{in}, V_{Pr})$ be a program. We associate the Büchi automaton $\mathcal{B}_{Pr} = (S, \rightsquigarrow, S_{in}, S)$ over the alphabet $\Sigma \times 2^P$ with Pr where $\rightsquigarrow$ is given by: $(s, (a, R), s') \in \rightsquigarrow$ iff $(s, a, s') \in \longrightarrow$ and $V_{Pr}(s) = R$.

Let α be a specification. Then we construct the Büchi automaton $\mathcal{B}_{\sim\alpha}$ corresponding to the *negation* of α. Let $\mathcal{B}_{\sim\alpha} = (Q, \Longrightarrow, Q_{in}, F)$. Recall that

each state in Q is of the form (A, x) where A is an atom. We now convert this automaton into the automaton $\widehat{\mathcal{B}} = (Q, \Rrightarrow, Q_{in}, F)$ over the alphabet $\Sigma \times 2^P$ by defining $\Rrightarrow$ as: $((A, x), (a, R), (B, y)) \in \Rrightarrow$ iff $((A, x), a, (B, y)) \in \Longrightarrow$ and $A \cap P = R$. Finally, let $\mathcal{B}$ be the Büchi automaton which accepts the intersection of the languages accepted by $\mathcal{B}_{Pr}$ and $\widehat{\mathcal{B}}$. It is straightforward to check that $Pr \models \alpha$ iff the language accepted by $\mathcal{B}$ is *empty*. An easy analysis of the size of $\mathcal{B}$ leads to:

Theorem 2.2 *The model checking problem for* LTL(Σ) *is decidable in time* $O(|Pr| \cdot 2^{|\alpha|})$.

In what follows, automata-theoretic constructions and expressiveness issues will play a considerable role. These topics can be treated in a simpler fashion if we eliminate atomic propositions. Most of the material we present can easily accomodate atomic propositions with some notational overhead. Hence from now on, we will not — except for some passing remarks — deal with atomic propositions. To be specific, the syntax of LTL(Σ) will be assumed to be:

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

Notice that a model is now just a member of Σ^ω with the semantics being the obvious one ($\top$ is always true). The set of models of a formula constitute a language of infinite words. More precisely, each α induces the language L_α given by:

$$L_\alpha = \{\sigma \mid \sigma, \varepsilon \models \alpha\}.$$

A program is now just a finite-state transition system $Pr = (S, \longrightarrow, S_{in})$ over Σ. Each such program Pr has the language L_{Pr} associated with it. This is just the language accepted by the Büchi automaton $(S, \longrightarrow, S_{in}, S)$. It is also easy to see that $Pr \models \alpha$ iff $L_{Pr} \subseteq L_\alpha$ iff $L_{Pr} \cap L_{\sim\alpha} = \emptyset$.

3 Mazurkiewicz Traces and Trace Consistent Properties

Here we wish to introduce the notion of traces from the standpoint of sequences. This will enable us to define the notion of a trace consistent property. This notion plays an important role in partial order based reducion methods. As pointed out in the introduction, it also provides the motivation for studying trace based linear time temporal logics.

A *(Mazurkiewicz) trace alphabet* is a pair (Σ, I), where Σ, the alphabet, is a finite set and $I \subseteq \Sigma \times \Sigma$ is an irreflexive and symmetric *independence*

relation. In most applications, Σ consists of the actions performed by a distributed system while I captures a static notion of causal independence between actions. The idea is that contiguous independent actions occur with no causal order between them. Thus, every sequence of actions from Σ corresponds to an interleaved observation of a partially-ordered stretch of system behaviour. This leads to a natural equivalence relation over execution sequences: two sequences are equated iff they correspond to different interleavings of the same partially-ordered stretch of behaviour.

For the rest of the section we fix a trace alphabet (Σ, I) and assume the terminology developed in the previous section for objects derived from Σ. We define $D = (\Sigma \times \Sigma) - I$ to be the *dependency relation*. Note that D is reflexive and symmetric. A set $p \subseteq \Sigma$ is called a *D-clique* iff $p \times p \subseteq D$. The equivalence relation $\approx_I \subseteq \Sigma^\infty \times \Sigma^\infty$ induced by I is given by:

$$\sigma \approx_I \sigma' \text{ iff } \sigma \restriction p = \sigma' \restriction p \text{ for every } D\text{-clique } p.$$

Here and elsewhere, if A is a finite set, $\rho \in A^\infty$ and $B \subseteq A$ then $\rho \restriction B$ is the sequence obtained by erasing from ρ all occurrences of letters in $A - B$.

Clearly $\approx_I$ is an equivalence relation. Notice that if $\sigma = \tau ab\sigma_1$ and $\sigma' = \tau ba\sigma_1$ with $(a, b) \in I$ then $\sigma \approx_I \sigma'$. Thus σ and σ' are identified if they differ only in the order of appearance of a pair of adjacent independent actions. In fact, for finite words, an alternative way to characterize $\approx_I$ is to say that $\sigma \approx_I \sigma'$ iff σ' can be obtained from σ by a finite sequence of permutations of adjacent independent actions. However the definition of $\approx_I$ in terms of permutations can not be directly transported to infinite words, which is why we work with the definition presented here.

The equivalence classes generated by $\approx_I$ are called *(Mazurkiewicz) traces*. A set of traces is called a *trace language*. The theory of traces is well developed and documented—see [6, 7] for basic material as well as a substantial number of references to related work.

A variety of models of distributed systems naturally have a trace alphabet associated with them [55]. It also turns out that many interesting properties of distributed systems respect the equivalence relation induced by these trace alphabets. This has important consequences for the practical verification of such properties.

The key notion in this context is that of a trace consistent property. To bring this out, we start with a trace alphabet (Σ, I) and recall the remarks concerning the abolition of atomic propositions at the end of Section 2. Let $L \subseteq \Sigma^\omega$. We say that L is *trace consistent* in case $\sigma \in L$ and $\sigma \approx_I \sigma'$ implies $\sigma' \in L$; for every $\sigma, \sigma' \in \Sigma^\omega$. In other words, either *all* members of a trace are in L or *none* of them are. We say that the formula α in LTL(Σ) is trace

consistent in case L_α is trace consistent. It is not hard to see that there is a one-to-one correspondence between trace languages and trace consistent languages of strings.

Now suppose Pr is a program over Σ which has a trace alphabet (Σ, I) associated with it in some natural manner. Suppose further that L_{Pr}, the linear time behaviour of Pr, is trace consistent (we will see a number of models of distributed programs that possess these features in the material to follow). Now consider a specification α which happens to be trace consistent. Then, as remarked at the end of Section 2, verifying $Pr \models \alpha$ boils down to verifying $L_{Pr} \subseteq L_\alpha$. Instead of checking $L_{Pr} \subseteq L_\alpha$ we can choose to check $L' \subseteq L_\alpha$ where L' is designed to be such that $L' \subseteq L_{Pr}$ and for every $\sigma \in L_{Pr}$, $[\sigma] \cap L' \neq \emptyset$. The key point is, the finite representation of L' can be often substantially smaller than the representation of Pr. This is the insight underlying many of the so called partial-order methods deployed in the model checking world [17, 35, 50].

As pointed out in the introduction this is also the main motivation for considering the trace-based linear time temporal logics that we will encounter later. We shall conclude this section with some examples.

Recall the material on elementary net systems introduced in Section 2. Suppose $\mathcal{N} = (B, E, F, c_{in})$ is an elementary net system. Each such system induces the independence relation $I_\mathcal{N}$ given by:

$$I_\mathcal{N} = \{(e_1, e_2) \mid ({}^\bullet e_1 \cup e_1^\bullet) \cap ({}^\bullet e_2 \cup e_2^\bullet) = \emptyset\}.$$

Let $e \in E$ and consider the formula $\Box\Diamond\langle e\rangle\top$. The property captured by this formula says that (along every computation) the event e occurs infinitely often. It is easy to see that this is a trace consistent property with respect to the trace alphabet $(E, I_\mathcal{N})$. Next consider the net system of Figure 1.

Consider the formula $\beta = \Box\Diamond(\langle e\rangle\top \wedge \langle e'\rangle\top)$. Suppose $\sigma = (e_1e_2ee')^\omega$ and $\sigma' = (e_1e'e_2e)^\omega$. Then $\sigma, \varepsilon \models \beta$ and $\sigma \approx_{I_\mathcal{N}} \sigma'$ but $\sigma', \varepsilon \not\models \beta$. Thus this property is not trace consistent with respect to the trace alphabet induced by this net system.

4 Product Languages and Automata

We will now exhibit a restricted but useful class of distributed behaviours that we call product behaviours. Such behaviours are generated by a network of sequential agents that coordinate their activities by performing common actions together. It will turn out that product behaviours are naturally trace consistent. They also constitute a clean and yet non-trivial subset of the class of trace behaviours considered later.

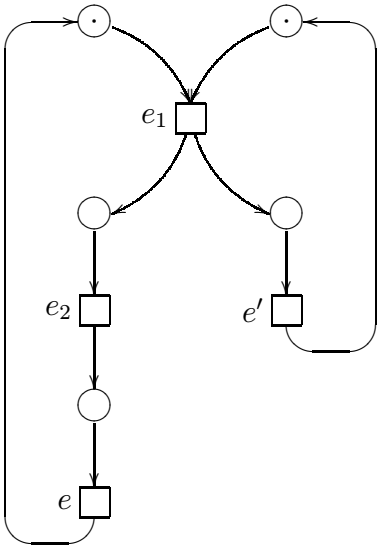

Figure 1: Example elementary net system

We first study product Büchi automata. We then formulate in Section 5 the product version of LTL(Σ). We will then use product Büchi automata to solve the satisfiability and model checking problems for the product version of LTL(Σ). The technical details — which we suppress here — can be found in [47]. The key notion underlying product behaviours is that of a distributed alphabet. It can be viewed as an "implementation" of a trace alphabet. As a result, distributed alphabets play a fundamental role in the automata-theoretic aspects of trace languages [15, 58]. This will become more clear when the material in Section 6 is encountered.

A *distributed alphabet* is a family $\{\Sigma_p\}_{p\in\mathcal{P}}$ where $\mathcal{P}$ is a finite non-empty set of agents (also referred to as processes in the sequel) and Σ_p is a finite non-empty alphabet for each $p \in \mathcal{P}$. The idea is that whenever an action from Σ_p occurs, the agent p must participate in it. Hence the agents can constrain each other's behaviour, both directly and indirectly.

Trace alphabets and distributed alphabets are closely related to each other. Let $\widetilde{\Sigma} = \{\Sigma_p\}_{p\in\mathcal{P}}$ be a distributed alphabet. Then $\Sigma_{\mathcal{P}}$, the global alphabet associated with $\widetilde{\Sigma}$, is the collection $\bigcup_{p\in\mathcal{P}} \Sigma_p$. The distribution of $\Sigma_{\mathcal{P}}$ over $\mathcal{P}$ can be described using a *location function* $\mathrm{loc}_{\widetilde{\Sigma}} : \Sigma_{\mathcal{P}} \to 2^{\mathcal{P}}$ defined as follows:

$$\mathrm{loc}_{\widetilde{\Sigma}}(a) = \{p \mid a \in \Sigma_p\}.$$

This in turn induces the relation $I_{\widetilde{\Sigma}} \subseteq \Sigma_{\mathcal{P}} \times \Sigma_{\mathcal{P}}$ given by:

$$(a, b) \in I_{\widetilde{\Sigma}} \text{ iff } \mathrm{loc}_{\widetilde{\Sigma}}(a) \cap \mathrm{loc}_{\widetilde{\Sigma}}(b) = \emptyset.$$

Clearly $I_{\widetilde{\Sigma}}$ is irreflexive and symmetric and hence $(\Sigma_{\mathcal{P}}, I_{\widetilde{\Sigma}})$ is a trace alphabet. Thus every distributed alphabet canonically induces a trace alphabet. Two actions are independent according to $\widetilde{\Sigma}$ if they are executed by disjoint sets of processes. Henceforth, we write loc for $\mathrm{loc}_{\widetilde{\Sigma}}$ whenever $\widetilde{\Sigma}$ is clear from the context.

Going in the other direction there are, in general, many different ways to implement a trace alphabet as a distributed alphabet. A standard approach is to create a separate agent for each maximal D-clique generated by (Σ, I). Recall that a D-clique of (Σ, I) is a non-empty subset $p \subseteq \Sigma$ such that $p \times p \subseteq D$. Let $\mathcal{P}$ be the set of maximal D-cliques of (Σ, I). This set of processes induces the distributed alphabet $\widetilde{\Sigma} = \{\Sigma_p\}_{p \in \mathcal{P}}$ where $\Sigma_p = p$ for every process p. The alphabet $\widetilde{\Sigma}$ implements (Σ, I) in the sense that the canonical trace alphabet induced by it is exactly (Σ, I). In other words, $\Sigma_{\mathcal{P}} = \Sigma$ and $I_{\widetilde{\Sigma}} = I$.

For example, consider the trace alphabet (Σ, I) where $\Sigma = \{a, b, d\}$ and $I = \{(a, b), (b, a)\}$. The canonical D-clique implementation of (Σ, I) yields the distributed alphabet $\widetilde{\Sigma} = \{\{a, d\}, \{d, b\}\}$.

Through the rest of the section we fix a distributed alphabet $\{\Sigma_p\}_{p \in \mathcal{P}}$ and set $\Sigma = \Sigma_{\mathcal{P}}$. It will be convenient to assume that $\mathcal{P} = \{1, 2, \ldots, K\}$. Further, the ith component of a K-tuple $x = (x_1, x_2, \ldots, x_K)$ will be written as $x[i]$. In other words, $x[i] = x_i$.

A *product Büchi automaton* over $\widetilde{\Sigma}$ is a structure $\mathcal{A} = (\{\mathcal{A}_i\}_{i=1}^{K}, Q_{in})$ where $\mathcal{A}_i = (Q_i, \longrightarrow_i, F_i, F_i^{\omega})$ for each i such that :

- Q_i is a 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.
- $F_i^{\omega} \subseteq Q_i$ is a set of infinitary accepting states.
- $Q_{in} \subseteq Q_1 \times Q_2 \times \cdots \times Q_K$ is a set of global initial states.

We use two types of accepting states for the components in order to be able to handle both finite and infinite behaviours. Even if one is interested only in global infinite behaviours, finite behaviours at the component level must be treated; a component might quit after engaging in a finite number of actions while a part of the network runs forever. We use global initial states to obtain the required expressive power. In general, the automaton will not be able to branch off into different parts of the state space, starting from a single global initial state. This will be brought out through a simple

example after we define the language behaviour of product automata. The same example will also illustrate why using the cartesian product of local initial state sets as global initial states will result in a loss of expressive power.

Let $\mathcal{A} = (\{\mathcal{A}_i\}_{i=1}^K, Q_{in})$ be a product Büchi automaton over $\widetilde{\Sigma}$. From now on we will say just "product automata". Also, we shall often suppress the mention of $\widetilde{\Sigma}$. We will also write $\{\mathcal{A}_i\}$ instead of $\{\mathcal{A}_i\}_{i=1}^K$. Let $\mathcal{A}_i = (Q_i, \longrightarrow_i, F_i, F_i^\omega)$. Then we set $Q_G^{\mathcal{A}} = Q_1 \times Q_2 \times \ldots \times Q_K$. When $\mathcal{A}$ is clear from the context, we will write Q_G instead of $Q_G^{\mathcal{A}}$. The global transition relation of $\mathcal{A}$ is denoted as $\longrightarrow_{\mathcal{A}}$ and it is the subset of $Q_G \times \Sigma \times Q_G$ given by:

$$q \xrightarrow{a}_{\mathcal{A}} q' \text{ iff } \forall\ i \in \mathrm{loc}(a) : q[i] \xrightarrow{a}_i q'[i] \text{ and } \forall\ i \notin \mathrm{loc}(a) : q[i] = q'[i].$$

Let $\sigma \in \Sigma^\infty$. A *run* of $\mathcal{A}$ over σ is a map $\rho : \mathrm{Prf}(\sigma) \longrightarrow Q_G$ which satisfies:

- $\rho(\varepsilon) \in Q_{in}$.
- $\forall\ \tau a \in \mathrm{prf}(\sigma).\ \rho(\tau) \xrightarrow{a}_{\mathcal{A}} \rho(\tau a)$.

A simple but useful property of runs is the following. Suppose ρ is a run of the product automaton $\mathcal{A}$ over σ. Further suppose that $\tau, \tau' \in \mathrm{Prf}(\sigma)$ such that $\tau \restriction i = \tau' \restriction i$ for some i. Then $\rho(\tau)[i] = \rho(\tau')[i]$.

Let ρ be a run of the product automaton $\mathcal{A}$ over σ. Then ρ is *accepting* iff for each i, the following condition is satisfied:

- If $\sigma \restriction i$ is finite then $\rho(\tau)[i] \in F_i$ where $\tau \in \mathrm{prf}(\sigma)$ such that $\tau \restriction i = \sigma \restriction i$.
- If $\sigma \restriction i$ is infinite then $\rho(\tau a)[i] \in F_i^\omega$ for infinitely many $\tau a \in \mathrm{prf}(\sigma)$ with $a \in \Sigma_i$.

If $\sigma \restriction i$ is finite then clearly there exists $\tau \in \mathrm{prf}(\sigma)$ such that $\tau \restriction i = \sigma \restriction i$. Now the above property of runs assures us that the notion of an accepting run is well-defined. In case $\sigma \restriction i$ is infinite the acceptance condition can also be phrased as:

- $\rho(\tau)[i] \in F_i^\omega$ for infinitely many $\tau \in \mathrm{prf}(\sigma)$.

This once again follows easily from the definition of a run. We now define $\mathcal{L}(\mathcal{A})$, the *language accepted by the product automaton* $\mathcal{A}$ as,

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

Now consider the alphabet $(\{a, d\}, \{d, b\})$ and the language $L = \{ad, bd\}$. Figure 2 shows a product automaton over this alphabet which accepts L. It

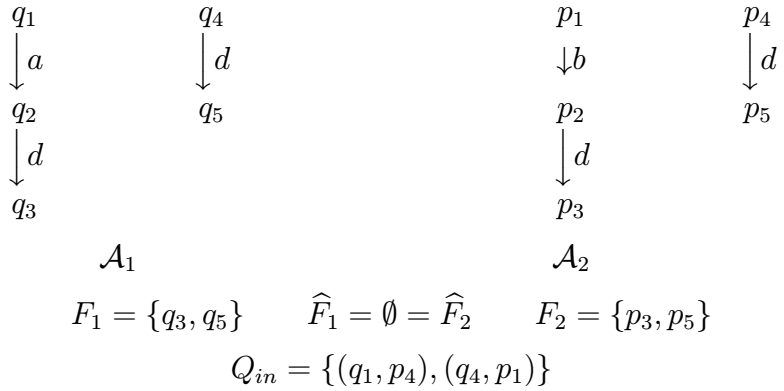

Figure 2: Product automaton accepting $L = \{ad, bd\}$

is easy to verify that *no* product automaton over this alphabet with a *single* global initial state can accept L. It is also easy to verify that no product automaton whose set of initial states is a cartesian product of component initial state sets can accept this language.

A crucial property of product automata is that they accept $\approx$-consistent languages.

Lemma 4.1 *Let $\mathcal{A} = (\{\mathcal{A}_i\}, Q_{in})$ be a product automaton over $\widetilde{\Sigma}$. Then $\mathcal{L}(\mathcal{A})$ is trace consistent.*

The class of languages accepted by product automata can now be characterized. To this end we define the K-ary operation $\otimes : 2^{\Sigma_1^\infty} \times 2^{\Sigma_2^\infty} \times \cdots \times 2^{\Sigma_K^\infty} \to 2^{\Sigma^\infty}$ via $\otimes(L_1, \ldots, L_K) = \{\sigma \mid \sigma \restriction i \in L_i \text{ for each } i\}$.

In what follows we will write $L = L_1 \otimes L_2 \cdots \otimes L_K$ to denote the fact $\otimes(L_1, \ldots, L_K) = L$. We say that $L \subseteq \Sigma^\infty$ is a *direct product language* over $\widetilde{\Sigma}$ iff $\exists\ L_i \subseteq \Sigma_i^\infty$ for each i such that $L = L_1 \otimes L_2 \otimes \cdots \otimes L_K$. Here are two useful properties of direct product languages. In stating this result and elsewhere we will say "product language" instead of "product language over $\widetilde{\Sigma}$" etc.

Proposition 4.2

1. *Let L be a direct product language and $\sigma \in \Sigma^\infty$. Then $\sigma \in L$ iff for each i there exists $\sigma_i \in L$ such that $\sigma \restriction i = \sigma_i \restriction i$.*

2. *Let $L \subseteq \Sigma^\infty$. Then L is a direct product language iff $L = \widehat{L}_1 \otimes \widehat{L}_2 \otimes \cdots \otimes \widehat{L}_K$ where $\widehat{L}_i = \{\sigma \restriction i \mid \sigma \in L\}$ for each i.*

As usual, for an alphabet Σ and $L \subseteq \Sigma^\infty$ we say that L is *regular* iff $L \cap \Sigma^*$ is a regular subset of Σ^* and $L \subseteq \Sigma^\omega$ is an ω-regular subset of Σ^ω as described in Section 2. We can now define the class of languages accepted by product automata.

Definition 4.3

- $\mathcal{R}_0^\otimes(\widetilde{\Sigma})$ is the subset of 2^{Σ^∞} given by $L \in \mathcal{R}_0^\otimes(\widetilde{\Sigma})$ iff $L = L_1 \otimes L_2 \otimes \cdots \otimes L_K$ with each L_i a regular subset of Σ_i^∞.
- $\mathcal{R}^\otimes(\widetilde{\Sigma})$ is the least subset of 2^{Σ^∞} which contains $\mathcal{R}_0^\otimes$ and is closed under finite unions.

The class $\mathcal{R}^\otimes(\widetilde{\Sigma})$ defined above will be called the *regular product languages* over $\widetilde{\Sigma}$. As usual, we shall often write $\mathcal{R}_0^\otimes$ instead of $\mathcal{R}_0^\otimes(\widetilde{\Sigma})$ and write $\mathcal{R}^\otimes$ instead of $\mathcal{R}^\otimes(\widetilde{\Sigma})$. An interesting observation concerning $\mathcal{R}^\otimes$ is the following:

Proposition 4.4 *$\mathcal{R}^\otimes$ is closed under boolean operations.*

It turns out that $\mathcal{R}^\otimes$ is precisely the class of languages accepted by product automata.

Theorem 4.5 ([47]) *Let $L \subseteq \Sigma^\infty$. Then $L \in \mathcal{R}^\otimes$ iff there exists a product automaton $\mathcal{A}$ such that $L = \mathcal{L}(\mathcal{A})$.*

We shall be using product automata to settle the decidability and model checking problems for the logic LTL$^\otimes$ to be introduced in the next section. In anticipation of this, we shall put down two more results concerning product automata. While doing so and elsewhere the *size* of the product automaton $\mathcal{A}$ will be understood to be $|Q_G|$.

Theorem 4.6 *Let $\mathcal{A}$ be a product automaton. Then the question $\mathcal{L}(\mathcal{A}) \stackrel{?}{=} \emptyset$ can be settled in time $O(2^{2K} \cdot n^2)$ where n is the size of $\mathcal{A}$.*

Theorem 4.7 *Let $\mathcal{A}^1$ and $\mathcal{A}^2$ be two product automata. Then one can effectively construct a product automaton $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}^1) \cap \mathcal{L}(\mathcal{A}^2)$ and moreover $n = O(2^K \cdot n_1 \cdot n_2)$ where n is the size of $\mathcal{A}$ and n_ℓ is the size of $\mathcal{A}^\ell$ for $\ell = 1, 2$.*

5 A Product Version of LTL

We now wish to design a product version of LTL denoted $\mathrm{LTL}^{\otimes}(\widetilde{\Sigma})$. 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 $a \in \Sigma_i$ and α is a formula with $loc(\alpha) \subseteq \{i\}$. Then $\langle a \rangle_i \alpha$ is a formula and $loc(\langle a \rangle_i \alpha) = \{i\}$.
- Suppose α and β are formulas such that $\mathrm{loc}(\alpha), \mathrm{loc}(\beta) \subseteq \{i\}$. Then $\alpha \mathcal{U}_i \beta$ is a formula. Moreover, $\mathrm{loc}(\alpha \mathcal{U}_i \beta) = \{i\}$.

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

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

Stated differently, the syntax of $\mathrm{LTL}_i^{\otimes}(\widetilde{\Sigma})$ is given inductively by:

- $\top \in \mathrm{LTL}_i^{\otimes}(\widetilde{\Sigma})$.
- If α and β are in $\mathrm{LTL}_i^{\otimes}(\widetilde{\Sigma})$ then $\sim\alpha$ and $\alpha \vee \beta$ are in $\mathrm{LTL}_i^{\otimes}(\widetilde{\Sigma})$.
- If α is in $\mathrm{LTL}_i^{\otimes}(\widetilde{\Sigma})$ and $a \in \Sigma_i$ then $\langle a \rangle_i \alpha$ is in $\mathrm{LTL}_i^{\otimes}(\widetilde{\Sigma})$.
- If α and β are in $\mathrm{LTL}_i^{\otimes}(\widetilde{\Sigma})$ then $\alpha \mathcal{U}_i \beta$ is in $\mathrm{LTL}_i^{\otimes}(\widetilde{\Sigma})$.

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

As before we derive some useful modalities:

- $O_i \alpha \stackrel{\Delta}{\Longleftrightarrow} \bigvee_{a \in \Sigma_i} \langle a \rangle_i \alpha$.
- $\Diamond_i \alpha \stackrel{\Delta}{\Longleftrightarrow} \top \mathcal{U}_i \alpha$.
- $\Box_i \alpha \stackrel{\Delta}{\Longleftrightarrow} \sim \Diamond_i \sim \alpha$.

Let $M = \sigma$ be a model and $\tau \in \mathrm{prf}(\sigma)$. The following assertions can now easily be checked.

- $\sigma, \tau \models O_i \alpha$ iff there exists $\tau' \in prf(\sigma)$ such that $\sigma, \tau' \models \alpha$ and $|\tau'_i| = |\tau_i| + 1$.
- $\sigma, \tau \models \Diamond_i \alpha$ iff there exists τ' with $\tau\tau' \in \mathrm{prf}(\sigma)$ such that $\sigma, \tau\tau' \models \alpha$.
- $\sigma, \tau \models \Box_i \alpha$ iff for each τ', $\tau\tau' \in \mathrm{prf}(\sigma)$ implies $\sigma, \tau\tau' \models \alpha$.

Note that $O_i \alpha$ is the i-local version of the usual next-state operator of LTL.

We will say that a formula $\alpha \in \mathrm{LTL}^{\otimes}(\widetilde{\Sigma})$ 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

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

We will show the satisfiability problem for $\mathrm{LTL}^{\otimes}(\widetilde{\Sigma})$ is solvable in deterministic exponential time. This will be achieved by effectively constructing a product automaton $\mathcal{A}_\alpha$ for each $\alpha \in \mathrm{LTL}^{\otimes}(\widetilde{\Sigma})$ such that the language accepted by $\mathcal{A}_\alpha$ is non-empty iff α is satisfiable. Our construction is a generalization of the one for LTL in Section 2. The solution to the satisfiability problem will at once lead to a solution to the model checking problem for programs modelled as a product of sequential agents.

Through the rest of the section we fix a formula $\alpha_0 \in \mathrm{LTL}^{\otimes}(\widetilde{\Sigma})$. As before we will for convenience assume that the derived local next-state modality O_i is included in the syntax of $\mathrm{LTL}^{\otimes}$. 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 $\alpha, \beta \in cl(\alpha_0)$.
- $\langle a \rangle_i \alpha \in cl(\alpha_0)$ implies $\alpha \in cl(\alpha_0)$.
- $\alpha \mathcal{U}_i \beta \in cl(\alpha_0)$ implies $\alpha, \beta \in cl(\alpha_0)$. In addition, $O_i(\alpha \mathcal{U}_i \beta) \in cl(\alpha_0)$.

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 $CL_i(\alpha_0) = CL(\alpha_0) \cap \mathrm{LTL}_i^{\otimes}$ 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 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 $\beta \in A$.
- $\alpha \mathcal{U}_i \beta \in A$ iff $\beta \in A$ or $\alpha, O_i(\alpha \mathcal{U}_i \beta) \in A$.

The set of i-type atoms is denoted AT_i. We next define, for each $\alpha \in CL(\alpha_0)$ and $(A_1, \ldots, A_K) \in AT_1 \times \ldots \times AT_K$, the predicate $\mathrm{Member}(\alpha, (A_1, \ldots, A_K))$. 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 $\gamma \in (A_1, \ldots, A_K)$.

Finally, we set $U_i = \{\alpha \mathcal{U}_i \beta \mid \alpha \mathcal{U}_i \beta \in CL_i(\alpha_0)\}$ for each i. The product automaton $\mathcal{A}_{\alpha_0}$ associated with α_0 is now defined to be $\mathcal{A}_{\alpha_0} = (\{\mathcal{A}_i\}, Q_{in})$ where, for each i, $\mathcal{A}_i = (Q_i, \longrightarrow_i, F_i, F_i^{\omega})$ is specified as follows:

- $Q_i = AT_i \times \{\mathsf{off}, \mathsf{on}\} \times 2^{U_i}$
- $\longrightarrow_i \subseteq Q_i \times \Sigma_i \times Q_i$ is given by, $(A, x, u) \xrightarrow{a}_i (B, y, v)$ iff the following conditions are met.

1. $x = \mathsf{on}$ and for all $\langle a\rangle_i\alpha \in CL_i(\alpha_0)$, $\langle a\rangle_i\alpha \in A$ iff $\alpha \in B$ and for all $O_i\alpha \in CL_i(\alpha_0)$, $O_i\alpha \in A$ iff $\alpha \in B$. Moreover, if $\langle b\rangle_i\beta \in A$ then $b = a$.
2. If $u \neq \emptyset$ then $v = \{\alpha\mathcal{U}_i\beta \mid \alpha\mathcal{U}_i\beta \in u \text{ and } \beta \notin B\}$. If $u = \emptyset$ then $v = \{\alpha\mathcal{U}_i\beta \mid \alpha\mathcal{U}_i\beta \in B \text{ and } \beta \notin B\}$.

- $F_i \subseteq Q_i$ is given by: $(A, x, u) \in F_i$ iff $x = \mathsf{off}$ and for all $\langle a\rangle_i\alpha \in CL_i(\alpha_0)$, $\langle a\rangle_i\alpha \notin A$ and for all $O_i\alpha \in CL_i(\alpha_0)$, $O_i\alpha \notin A$.
- $F_i^\omega \subseteq Q_i$ is given by: $(A, x, u) \in F_i^\omega$ iff $u = \emptyset$.
- $Q_{in} \subseteq Q_1 \times Q_2 \times \ldots \times Q_K$ is given by: $((A_1, x_1, u_1), \ldots, (A_K, x_K, u_K)) \in Q_{in}$ iff $\alpha_0 \in (A_1, \ldots, A_K)$ and $u_i = \emptyset$ for every i.

It is not difficult to now establish the next result by an application of Theorem 4.6.

Theorem 5.1 *α_0 is satisfiable iff $\mathcal{L}(\mathcal{A}_{\alpha_0}) \neq \emptyset$. Hence the satisfiability problem for* LTL$^\otimes$ *is decidable in exponential time.*

We now turn to the model checking problem for LTL$^\otimes$. A product program (over $\widetilde{\Sigma}$) is a structure $Pr = (\{Pr_i\}_{i=1}^K, Q_{in}^{Pr})$ where, for each i, $Pr_i = (Q_i, \longrightarrow_i)$ with Q_i a finite set and $\longrightarrow_i \subseteq Q_i \times \Sigma_i \times Q_i$. Since we have agreed to drop atomic propositions there is no need for (local) interpretations for the atomic propositions. Let us further assume for convenience that Q_{in}^{Pr} is a singleton with q_{in} as its sole member and with $q_{in}[i] = q_{in}^i$ for each i. With each such program we can associate the product automaton $\mathcal{A}_{Pr} = (\{\mathcal{A}_i\}_{i=1}^K, \{q_{in}\})$ where $\mathcal{A}_i = (Q_i, \longrightarrow_i, Q_i, Q_i)$ for each i.

Now let Pr be a product program and α_0 be a formula of LTL$^\otimes$. As in the case for LTL, we say that Pr *meets the specification* α_0 — again denoted $Pr \models \alpha_0$ — iff $\sigma, \varepsilon \models \alpha_0$ for every $\sigma \in \mathcal{L}(\mathcal{A}_{Pr})$. Once again, using Theorem 4.7 it is not difficult to prove the following.

Theorem 5.2 *The model checking problem for* LTL$^\otimes$ *is decidable in time* $O(|Pr| \cdot 2^{|\alpha_0|})$.

We wish to observe that each product program can be represented as a Σ-labelled 1-safe net system. To see this let $Pr = (\{Pr_i\}_{i=1}^K, \{q_{in}\})$ be a product program. Let's assume without loss of generality that the family of local states $\{Q_i\}$ is pairwise disjoint. We set $Q = \bigcup_{i\in\mathcal{P}} Q_i$ and define an a-state to be a map $q_a : \mathrm{loc}(a) \to Q$ which satisfies $q_a(i) \in Q_i$ for each i in $\mathrm{loc}(a)$. (A more elaborate development of these notions will appear in the next section).

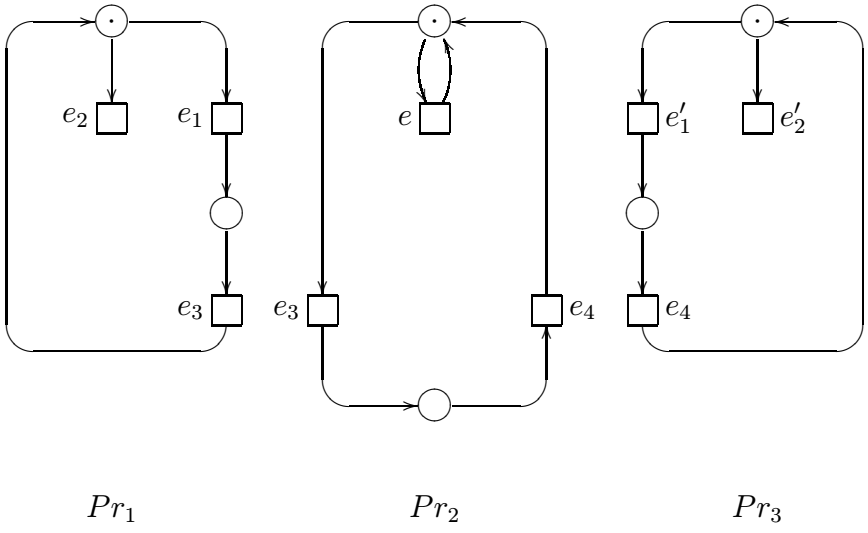

Figure 3: 1-safe net with three components

An a-event is a pair of a-states (q_a, q'_a) which satisfies $q_a(i) \xrightarrow{a}_i q'_a(i)$ for each i in $\mathrm{loc}(a)$. We let E_a be the set of a-events. We can now define the Σ-labelled 1-safe net system representing Pr to be $\mathcal{N} = (B, E, F, c_{in}, \phi)$ where:

- $B = Q$
- $E = \bigcup_{a \in \Sigma} E_a$
- Let $q_i \in Q_i$ and $e = (q_a, q'_a) \in E_a$. Then $(q_i, e) \in F$ iff $i \in \mathrm{loc}(a)$ and $q_a(i) = q_i$. Similarly $(e, q_i) \in F$ iff $i \in \mathrm{loc}(a)$ and $q'_a(i) = q_i$.
- Let $e \in E$. Then $\phi(e) = a$ iff e is an a-event.

On the other hand each 1-safe net system which is covered by a set of S-components can be viewed as a (deterministic) product program; the alphabet of each component is its set of events. If necessary, S-complementation can be performed to ensure that the system is covered by a set of S-components. We do not wish to enter into details here. Instead we show on Figure 3 an example of a 1-safe net system composed out of three components.

Let Pr denote the associated product program over the distributed alphabet $\{\{e_1, e_2, e_3\}, \{e_3, e_4\}, \{e'_1, e'_2, e_4\}\}$. Then it is easy to check that

$$Pr \models \Box_1 O_1 \top \supset \Box_3 O_3 \top.$$

This property says that along every computation, if the first component executes infinitely often then so does the third component. The point to note is that the first component and the third component do not have any common events and hence there is no direct communication between them. Nevertheless through the power of the boolean connectives alone the logic can make assertions about the way components that are "far apart" are required to influence each other's behaviour.

6 Trace Languages and Automata

Traces have many equivalent representations. Here we shall view them as restricted Σ-labelled partial orders. Abusing terminology we shall call these objects also *traces*. We will then argue that these objects are in a rather precise sense the same as the objects called traces defined in Section 3 in terms of equivalence classes of sequences.

Let T be a Σ-labelled poset. In other words, $(E, \leq)$ is a poset and $\lambda : E \to \Sigma$ is a labelling function. For $Y \subseteq E$ we define $\downarrow Y = \{x \mid \exists y \in Y : \ x \leq y\}$ and $\uparrow Y = \{x \mid \exists y \in Y : \ y \leq x\}$. In case $Y = \{y\}$ is a singleton we shall write $\downarrow y$ ($\uparrow y$) instead of $\downarrow\{y\}$ ($\uparrow \{y\}$). We also let $\lessdot$ be the relation: $x \lessdot y$ iff $x < y$ and for all $z \in E$, $x \leq z \leq y$ implies $x = z$ or $z = y$.

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

(T1) $\forall e \in E.$ $\quad \downarrow e$ is a finite set
(T2) $\forall e, e' \in E.$ $\ e \lessdot e'$ implies $\lambda(e)\ D\ \lambda(e')$.
(T3) $\forall e, e' \in E.$ $\ \lambda(e)\ D\ \lambda(e')$ implies $e \leq e'$ or $e' \leq e$.

We shall refer to members of E as *events*. The trace $T = (E, \leq, \lambda)$ is said to be *finite* if E is a finite set. Otherwise it is an *infinite* trace. Note that E is always a countable set. T is said to be *non-empty* in case $E \neq \emptyset$. We let $TR^{fin}(\Sigma, I)$ be the set of finite traces and $TR^{\omega}(\Sigma, I)$ be the set of infinite traces over (Σ, I) and set $TR(\Sigma, I) = TR^{fin}(\Sigma, I) \cup TR^{\omega}(\Sigma, I)$. Often we will write TR^{fin} instead of $TR^{fin}(\Sigma, I)$ etc. As before, a subset of traces $L_{Tr} \subseteq TR$ will be called a *trace language*.

Let $T = (E, \leq, \lambda)$ be a trace. The finite prefixes of T, to be called configurations, will play a crucial role in what follows. A *configuration* of T is a finite subset $c \subseteq E$ such that $c = \downarrow c$. We let $\mathcal{C}_T$ be the set of configurations of T and let c, c', c'' range over $\mathcal{C}_T$. Note that $\emptyset$, the empty set, is a configuration and $\downarrow e$ is a configuration for every $e \in E$. Finally, the transition relation $\longrightarrow_T \subseteq \mathcal{C}_T \times \Sigma \times \mathcal{C}_T$ is given by: $c \xrightarrow{a}_T c'$ iff there exists $e \in E$ such that $\lambda(e) = a$ and $e \notin c$ and $c' = c \cup \{e\}$. It is easy to see that if $c \xrightarrow{a}_T c'$ and $c \xrightarrow{a}_T c''$ then $c' = c''$.

Note that we have now introduced two different notions of traces; one in terms of equivalence classes of strings as in Section 3 and the other in terms of Σ-labelled partial orders as in this section. We now sketch briefly the constructions that show that $\Sigma^\infty/\approx_I$ and $TR(\Sigma, I)$ represent the same class of objects. We shall construct representation maps $\mathsf{str} : \Sigma^\infty/\approx_I \to TR(\Sigma, I)$ and $\mathsf{trs} : TR(\Sigma, I) \to \Sigma^\infty/\approx_I$ and state some results which show that these maps are "inverses" of each other. We shall not prove these results. The details can be easily obtained using the constructions developed in [55] for relating traces and event structures.

Henceforth, we will not distinguish between isomorphic elements in $TR(\Sigma, I)$. In other words, whenever we write $T = T'$ for traces $T = (E, \leq, \lambda)$ and $T' = (E', \leq', \lambda')$, we mean that there is a label-preserving isomorphism between T and T'.

Recall that for $\sigma \in \Sigma^\infty$, $[\sigma]$ stands for the $\approx_I$-equivalence class containing σ. We now define $\mathsf{str} : \Sigma^\infty \to TR(\Sigma, I)$. Let $\sigma \in \Sigma^\infty$. Then $\mathsf{str}(\sigma) = (E, \leq, \lambda)$ where:

- $E = \{\tau a \mid \tau a \in \mathrm{prf}(\sigma)\}$. Recall that $\tau \in \Sigma^*$ and $a \in \Sigma$. Thus $E = \mathrm{prf}(\sigma) - \{\varepsilon\}$, where ε is the null string.
- $\leq \subseteq E \times E$ is the least partial order which satisfies: For all $\tau a, \tau' b \in E$, if $\tau a \preceq \tau' b$ and $(a, b) \in D$ then $\tau a \leq \tau' b$.
- For $\tau a \in E$, $\lambda(\tau a) = a$.

The map str induces a natural map str' from $\Sigma^\infty/\approx_I$ to $TR(\Sigma, I)$ defined by $\mathsf{str}'([\sigma]) = \mathsf{str}(\sigma)$. One can show that if $\sigma, \sigma' \in \Sigma^\infty$, then $\sigma \approx_I \sigma'$ iff $\mathsf{str}(\sigma) = \mathsf{str}(\sigma')$. This observation guarantees that str' is well-defined. In fact, henceforth we shall write str to denote both str and str'.

Next, let $T = (E, \leq, \lambda) \in TR(\Sigma, I)$. Then $\sigma \in \Sigma^\infty$ is a *linearization* of T iff there exists a map $\rho : \mathrm{prf}(\sigma) \to \mathcal{C}_T$, such that the following conditions are met:

- $\rho(\varepsilon) = \emptyset$.
- $\forall \tau a \in \mathrm{prf}(\sigma)$ with $\tau \in \Sigma^*$, $\rho(\tau) \xrightarrow{a}_T \rho(\tau a)$.
- $\forall e \in E\ \exists \tau \in \mathrm{prf}(\sigma).\ e \in \rho(\tau)$.

The function ρ will be called a *run map* of the linearization σ. Note that the run map of a linearization is unique. In what follows, we shall let $\mathrm{lin}(T)$ to be the set of linearizations of the trace T.

We can now define the map $\mathsf{trs} : TR(\Sigma, I) \to \Sigma^\infty/\approx_I$ as: $\mathsf{trs}(T) = \mathrm{lin}(T)$. One can now show that for every $\sigma \in \Sigma^\infty$, $\mathsf{trs}(\mathsf{str}(\sigma)) = [\sigma]$ and for every $T \in TR(\Sigma, I)$, $\mathsf{str}(\mathsf{trs}(T)) = T$. This justify our claim that $\Sigma^\infty/\approx_I$ and $TR(\Sigma, I)$ are indeed two equivalent ways of talking about the same class of objects.

We note that every trace consistent subset L of Σ^∞ defines a trace language L_{Tr} given by $L_{Tr} = \{\mathsf{str}(\sigma) \mid \sigma \in L\}$ which has the property $\mathsf{trs}(L_{Tr}) = L$. In this sense every product language defines a trace language. We say that a trace language L_{Tr} is *regular* iff $\mathsf{trs}(L_{Tr})$ is a regular subset of Σ^∞. As we will see later not every (regular) trace language is a (regular) product language. Hence in order to recognize regular trace languages one will have to use strengthened versions of product automata. Such automata called asynchronous automata were formulated by Zielonka for recognizing regular languages of finite traces. These were then generalized for handling infinite traces by Gastin and Petit [15]. We will use a combination of these two types of automata for solving the satisfiability and model checking problems for the trace-based temporal logic called TrPTL to be considered in the next section.

Let $\widetilde{\Sigma}$ be a distributed alphabet with $\mathcal{P}$ as the associated set of agents. In an asynchronous automaton, each process $p \in \mathcal{P}$ is equipped with a finite non-empty set of local p-states, denoted S_p. It will be convenient to develop some notations for talking about "more global" states before defining these automata.

First we set $S = \bigcup_{p \in \mathcal{P}} S_p$ and call S the set of *local states*. We let P, Q range over non-empty subsets of $\mathcal{P}$ and let p, q range over $\mathcal{P}$. A Q-state is a map $s : Q \to S$ such that $s(q) \in S_q$ for every $q \in Q$. We let S_Q denote the set Q-states. We call $S_{\mathcal{P}}$ the set of *global states*.

We use a to abbreviate $\mathrm{loc}(a)$ when talking about states (recall that $\mathrm{loc}(a) = \{\ p \mid a \in \Sigma_p\ \}$). Thus an a-state is just a $\mathrm{loc}(a)$-state and S_a denotes the set of all $\mathrm{loc}(a)$-states.

A *distributed transition system* TS over $\widetilde{\Sigma}$ is a structure

$$(\{S_p\}, \{\longrightarrow_a\}, S_{in}),$$

where

- S_p is a finite non-empty set of p-states for each process p.
- For $a \in \Sigma$, $\longrightarrow_a \subseteq S_a \times S_a$ is a transition relation between a-states.
- $S_{in} \subseteq S_{\mathcal{P}}$ is a set of initial global states.

The idea is that an a-move by TS involves only the local states of the agents which participate in the execution a. This is reflected in the global transition relation $\longrightarrow_{TS} \subseteq S_{\mathcal{P}} \times \Sigma \times S_{\mathcal{P}}$ which is defined as follows: Suppose s and s' are two global states and s_a and s'_a are the two corresponding a-states. In other words, $s_a(i) = s(i)$ and $s'_a(i) = s'(i)$ for each i in $\mathrm{loc}(a)$. Then

$$s \xrightarrow{a}_{TS} s' \text{ iff } (s_a, s'_a) \in \longrightarrow_a \text{ and } s(j) = s'(j) \text{ for every } j \notin \mathrm{loc}(a).$$

From the definition of $\longrightarrow_{TS}$, it is clear that actions which are executed by disjoint sets of agents are processed independently by TS.

An *asynchronous automaton* over $\widetilde{\Sigma}$ is then a distributed transition system equipped with a set of global accepting states. More precisely, it is a structure $\mathcal{A} = (\{S_p\}, \{\longrightarrow_a\}, S_{in}, F)$ where

- $F \subseteq S_{\mathcal{P}}$ is a set of accepting global states.

A *trace run* of $\mathcal{A}$ over the finite trace $T = (E, \leq, \lambda)$ is a map $\rho : \mathcal{C}_T \to S_{\mathcal{P}}$ such that $\rho(\emptyset) \in S_{in}$ and for every $(c, a, c') \in \longrightarrow_T$, $\rho(c) \xrightarrow{a}_{TS} \rho(c')$. We say that ρ is an *accepting run* whenever $\rho(E) \in F$. The *language of finite traces accepted by* $\mathcal{A}$ is given by

$$\mathcal{L}_{Tr}(\mathcal{A}) = \{\ T \in TR^{fin} \mid \exists \text{ an accepting run of } \mathcal{A} \text{ over } T\ \}.$$

In the present setting Zielonka's fundamental result can now be formulated as

Theorem 6.1 ([58]) *$L \subseteq TR^{fin}(\Sigma, I)$ is regular iff $L = L_{Tr}(\mathcal{A})$ for some asynchronous automaton $\mathcal{A}$ over some $\widetilde{\Sigma}$ where $\widetilde{\Sigma}$ is a distributed alphabet whose induced trace alphabet is (Σ, I). Further, one may assume $\mathcal{A}$ to be deterministic and one may assume $\widetilde{\Sigma}$ to be the distributed alphabet induced by the maximal D-cliques of (Σ, I).*

This result has been generalized to the set of ω-regular trace languages by Gastin and Petit [15] in terms of asynchronous automata with Büchi acceptance conditions. Since we will treat both finite and infinite traces on an equal footing we will present a class of automata capable of accepting both finite and infinite traces. Hence our automata are essentially distributed transition systems augmented with *both* finite and infinite accepting states.

An *asynchronous Büchi automaton* over $\widetilde{\Sigma}$ is a structure

$$\mathcal{A} = (\{S_p\}, \{\longrightarrow_a\}, S_{in}, \{(F_p, F_p^{\omega})\}),$$

where:

- $(\{S_p\}, \{\longrightarrow_a\}, S_{in})$ is a distributed transition system.
- $F_p \subseteq S_p$ is a set of local finitary accepting states of process p.
- $F_p^\omega \subseteq S_p$ is a set of local infinitary accepting states of process p.

For convenience we will from now on denote this class of automata just "asynchronous automata".

To define acceptance we must now compute $\mathrm{Inf}_p(\rho)$, the set of p-states that are encountered infinitely often along ρ. When incorporating both finite and infinite behaviour in this richer domain we have to take care in defining the set of infinitely occuring states of process p. The obvious definition, namely $\mathrm{Inf}_p(\rho) = \{s_p \mid \rho(c)(p) = s_p \text{ for infinitely many } c \in \mathcal{C}_T\}$, will not work. The complication arises because some processes may make only finitely many moves, even though the overall trace consists of an infinite number of events.

For instance, consider the distributed alphabet $\widetilde{\Sigma}_0 = \{\{a\}, \{b\}\}$. In the corresponding distributed transition system, there are two processes p and q which execute a's and b's completely independently. Consider the trace $T = (E, \leq, \lambda)$ where $|E_p| = 1$ and E_q is infinite — i.e., all the infinite words in $\mathsf{trs}(T)$ contain one a and infinitely many b's. Let s_p be the state of p after executing a. Then, there will be infinitely many configurations whose p-state is s_p, even though p only moves a finite number of times.

Continuing with the same example, consider another infinite trace $T' = (E', \leq', \lambda')$ over the same alphabet where both E_p and E_q are infinite. Once again, let s_p be the local state of p after reading one a. Further, let us suppose that after reading the second a, p never returns to the state s_p. It will still be the case that there are infinitely many configurations whose p-state is s_p: consider the configurations $c_0, c_1, c_2, \ldots$ where c_j is the finite configuration after one a and j b's have occurred.

So, we have to define $\mathrm{Inf}_p(\rho)$ so as to detect whether or not process p is making progress. The appropriate formulation is as follows:

- *E_p is finite:* $\mathrm{Inf}_p(\rho) = \{s_p\}$, where $\rho(\downarrow E_p) = s$ and $s_p = s(p)$.
- *E_p is an infinite set:* $\mathrm{Inf}_p(\rho) = \{s_p \mid \text{for infinitely many } e \in E_p, s_e(p) = s_p, \text{ where } \rho(\downarrow e) = s_e\}$.

A *trace run* of an asynchronous automaton over the (possibly infinite) trace $T = (E, \leq, \lambda) \in TR$ is now defined in the obvious way. A run ρ of $\mathcal{A}$ over the (possibly infinite) trace $T = (E, \leq, \lambda)$ is *accepting* iff for each process p the following conditions are met:

- If E_p is *finite* then $\mathrm{Inf}_p(\rho) \cap F_p \neq \emptyset$.
- If E_p is *infinite* then $\mathrm{Inf}_p(\rho) \cap F_p^\omega \neq \emptyset$.

We then have the following characterization extending Theorem 6.1.

Theorem 6.2 *A trace language $L \subseteq TR(\Sigma, I)$ is regular iff $L = L_{Tr}(\mathcal{A})$ for an asynchronous automaton over $\widetilde{\Sigma}$ where $\widetilde{\Sigma}$ is a distributed alphabet whose induced trace alphabet is (Σ, I).*

It should be noted however that deterministic automata no longer suffice for accepting *all* regular languages.

We say that $\mathcal{A}$ is *in standard form* if

- For each p, $F_p \cap F_p^\omega = \emptyset$.
- For each $(s_a, t_a) \in \longrightarrow_a$ and $p \in \mathrm{loc}(a)$ we have that $s_a(p) \notin F_p$.

Thus, $\mathcal{A}$ is in standard form if the p-states in F_p are all "dead" and disjoint from F_p^ω. It is easy to convert every asynchronous automaton into standard form. All our asynchronous automata will be in standard form.

We conclude with a result concerning the emptiness problem for asynchronous automata.

Proposition 6.3 ([30]) *Let $\mathcal{A}$ be an asynchronous automaton in standard form. The emptiness problem is decidable in time $O(n^{2|\mathcal{P}|})$, where n is the largest of the local state spaces, S_p.*

We have defined here the languages defined by asynchronous automata in terms of traces. We note that these automata can be viewed — and this is the conventional approach — as automata running over Σ-sequences. Using the global transition relations of these automata one can easily define the string languages accepted by these automata. These languages will be naturally trace consistent w.r.t. the trace alphabets induced by the associated distributed alphabets. The resulting trace languages will be precisely the trace languages accepted by these automata according to the definitions we have provided here.

7 TrPTL

We present here the linear time temporal logic over traces called TrPTL. This is the first such logic patterned after PTL (i.e. LTL) formulated for traces. For a detailed treatment of this logic the reader is referred to [44, 45].

As before, it will be notationally convenient to deal with distributed alphabets in which the names of the processes are positive integers. Through this section and the next, we fix a distributed alphabet $\widetilde{\Sigma} = \{\Sigma_i\}_{i\in\mathcal{P}}$ with $\mathcal{P} = \{1, 2, \ldots, K\}$ and $K \geq 1$. We let i, j and k range over $\mathcal{P}$. As before, let P, Q range over non-empty subsets of $\mathcal{P}$. The trace alphabet induced by $\widetilde{\Sigma}$ is denoted (Σ, I). We assume the terminology and notations developed in the previous sections. In particular, when dealing with a $\mathcal{P}$-indexed family $\{X_i\}_{i\in\mathcal{P}}$ we will often write just $\{X_i\}$.

The logic TrPTL is parameterized by the class of distributed alphabets. Having fixed $\widetilde{\Sigma}$ we shall often almost always write TrPTL to mean TrPTL($\widetilde{\Sigma}$), the logic associated with $\widetilde{\Sigma}$. In order to better illustrate the main features of the logic we will first include atomic propositions. They will be dropped once we return to considering the technical aspects of the logic. We fix a finite non-empty set of atomic propositions P with p, q ranging over P. Then $\Phi_{\mathrm{TrPTL}(\widetilde{\Sigma})}$, the set of formulas of TrPTL($\widetilde{\Sigma}$), is defined inductively via:

- For $p \in P$ and $i \in \mathcal{P}$, $p(i)$ is a formula (which is to be read "p at i").
- If α and β are formulas, so are $\sim\alpha$ and $\alpha \vee \beta$.
- If α is a formula and $a \in \Sigma_i$ then $\langle a\rangle_i\alpha$ is a formula.
- If α and β are formulas so is $\alpha\mathcal{U}_i\beta$.

Throughout this section, we denote $\Phi_{\mathrm{TrPTL}(\widetilde{\Sigma})}$ as just Φ. In the semantics of the logic, which will be based on infinite traces, the i-view of a configuration will play a crucial role. Let $T \in TR^\omega$ with $T = (E, \leq, \lambda)$. Recall that $E_i = \{e \mid e \in E \text{ and } \lambda(e) \in \Sigma_i\}$. Let $c \in \mathcal{C}_T$ and $i \in \mathcal{P}$. Then $\downarrow^i(c)$ is the i-view of c and it is defined as:

$$\downarrow^i(c) = \downarrow(c \cap E_i).$$

We note that $\downarrow^i(c)$ is also a configuration. It is the "best" configuration that the agent i is aware of at c. We say that $\downarrow^i(c)$ is an *i-local configuration*. Let $\mathcal{C}_T^i = \{\downarrow^i(c) \mid c \in \mathcal{C}_T\}$ be the set of i-local configurations. For $Q \subseteq \mathcal{P}$ and $c \in \mathcal{C}_T$, we let $\downarrow^Q(c)$ denote the set $\bigcup\{\downarrow^i(c) \mid i \in Q\}$. Once again, $\downarrow^Q(c)$ is a configuration. It represents the collective knowledge of the processes in Q about the configuration c.

The following basic properties of traces follow directly from the definitions.

Proposition 7.1 *Let $T = (E, \leq, \lambda)$ be an infinite trace. The following statements hold.*

1. *Let* $\leq_i = \leq \cap (E_i \times E_i)$. *Then* $(E_i, \leq_i)$ *is a linear order isomorphic to* ω *if* E_i *is infinite and isomorphic to a finite initial segment of* ω *if* E_i *is finite.*

2. $(\mathcal{C}_T^i, \subseteq)$ *is a linear order. In fact* $(\mathcal{C}_T^i - \{\emptyset\}, \subseteq)$ *is isomorphic to* $(E_i, \leq_i)$.

3. *Suppose* $\downarrow^i(c) \neq \emptyset$ *where* $c \in \mathcal{C}_T$. *Then there exists* $e \in E_i$ *such that* $\downarrow^i(c) = \downarrow e$. *In fact* e *is the* $\leq_i$*-maximum event in* $(c \cap E_i)$.

4. *Suppose* $Q \subseteq Q' \subseteq \mathcal{P}$ *and* $c \in \mathcal{C}_T$. *Then* $\downarrow^Q(c) = \downarrow^Q(\downarrow^{Q'}(c))$. *In particular, for a single process* i, $\downarrow^i(c) = \downarrow^i(\downarrow^i(c))$.

We can now present the semantics of TrPTL. A model is a pair $M = (T, \{V_i\}_{i \in p})$ where $T = (E, \leq, \lambda) \in TR^\omega$ and $V_i : \mathcal{C}_T^i \to 2^P$ is a valuation function which assigns a set of atomic propositions to i-local configurations for each process i. Let $c \in \mathcal{C}_T$ and $\alpha \in \Phi$. Then $M, c \models \alpha$ denotes that α is satisfied at c in M and it is defined inductively as follows:

- $M, c \models p(i)$ for $p \in P$ iff $p \in V_i(\downarrow^i(c))$.
- $M, c \models \sim\alpha$ iff $M, c \not\models \alpha$.
- $M, c \models \alpha \vee \beta$ iff $M, c \models \alpha$ or $M, c \models \beta$.
- $M, c \models \langle a \rangle_i \alpha$ iff there exists $e \in E_i - c$ such that $\lambda(e) = a$ and $M, \downarrow e \models \alpha$. Moreover, for every $e' \in E_i$, $e' < e$ iff $e' \in c$.
- $M, c \models \alpha \mathcal{U}_i \beta$ iff there exists $c' \in \mathcal{C}_T$ such that $c \subseteq c'$ and $M, \downarrow^i(c') \models \beta$. Moreover, for every $c'' \in \mathcal{C}_T$, if $\downarrow^i(c) \subseteq \downarrow^i(c'') \subset \downarrow^i(c')$ then $M, \downarrow^i(c'') \models \alpha$.

Thus TrPTL is an action based multi-agent version of LTL. Indeed both in terms of its syntax and semantics, LTL(Σ) corresponds to the case where there is only one agent. The semantics of TrPTL when specialized down to this case yields the previous LTL(Σ) semantics.

Returning to TrPTL, the assertion $p(i)$ says that the i-view of c satisfies the atomic proposition p. Observe that we could well have $p(i)$ satisfied at c but not $p(j)$ (with $i \neq j$). It is interesting to note that all atomic assertions (that we know of) concerning distributed behaviours are local in nature. Indeed, it is well-known that global atomic propositions will at once lead to an undecidable logic in the current setting [25, 36].

Suppose $M = (T, \{V_i\})$ is a model and $c \xrightarrow{a}_T c'$ with $j \notin \mathrm{loc}(a)$. Then $M, c \models p(j)$ iff $M, c' \models p(j)$. In this sense the valuation functions are local.

There are, of course, a number of equivalent ways of formulating this idea which we will not get into here.

The assertion $\langle a\rangle_i\alpha$ says that the agent i will next participate in an a-event. Moreover, at the resulting i-view, the assertion α will hold. The assertion $\alpha\mathcal{U}_i\beta$ says that there is a future i-view (including the present i-view) at which β will hold and for all the intermediate i-views (if any) starting from the current i-view, the assertion α will hold.

Before considering examples of TrPTL specifications, we will introduce some notation. We let α, β with or without subscripts range over Φ. Abusing notation, we will use loc to denote the map which associates a set of *locations* with each formula.

- $\mathrm{loc}(p(i)) = \mathrm{loc}(\langle a\rangle_i\alpha) = \mathrm{loc}(\alpha\mathcal{U}_i\beta) = \{i\}$.
- $\mathrm{loc}(\sim\alpha) = \mathrm{loc}(\alpha)$.
- $\mathrm{loc}(\alpha \vee \beta) = \mathrm{loc}(\alpha) \cup \mathrm{loc}(\beta)$.

In what follows, $\Phi^i = \{\alpha \mid \mathrm{loc}(\alpha) = \{i\}\}$ is the set of i-type formulas. We note that unlike LTL$^{\otimes}$, a TrPTL formula of the form $\langle a\rangle_i\alpha$ could have $j \in \mathrm{loc}(\alpha)$ with $j \neq i$. A similar remark applies to the indexed until-operators.

A basic observation concerning the semantics of TrPTL can be phrased as follows:

Proposition 7.2 *Let $M = (T, \{V_i\})$ be a model, $c \in \mathcal{C}_T$ and α a formula such that $\mathrm{loc}(\alpha) \subseteq Q$. Then $M, c \models \alpha$ iff $M, \downarrow^Q(c) \models \alpha$.*

A corollary to this result is that in case $\alpha \in \Phi^i$ then $M, c \models \alpha$ if and only if $M, \downarrow^i(c) \models \alpha$. As a result, the formulas in Φ^i can be used in exactly the same manner as one would use LTL$^{\otimes}$ to express properties of the agent i. Boolean combinations of such local assertions can be used to capture various interaction patterns between the agents implied by the logical connectives as well as the coordination enforced by the distributed alphabet $\widetilde{\Sigma}$. For writing specifications, apart from the usual derived connectives that we already introduced in Section 2 for LTL, the following operators are also available:

- $\top \stackrel{\Delta}{\Longleftrightarrow} p_1(1) \vee \sim p_1(1)$ denotes the constant "True", where $P = \{p_1, p_2, \ldots\}$. We use $\bot = \sim\top$ to denote "False".
- $\Diamond_i\alpha \stackrel{\Delta}{\Longleftrightarrow} \top\mathcal{U}_i\alpha$ is a local version of the $\Diamond$ modality of LTL.
- $\Box_i\alpha \stackrel{\Delta}{\Longleftrightarrow} \sim\Diamond_i \sim\alpha$ is a local version of the $\Box$ modality of LTL.

- Let $X \subseteq \Sigma_i$ and $\overline{X} = \Sigma_i - X$. Then $\alpha \mathcal{U}_i^X \beta \stackrel{\Delta}{\Longleftrightarrow} (\alpha \wedge \bigwedge_{a \in \overline{X}} [a]_i \bot) \mathcal{U}_i \beta$. In other words $\alpha \mathcal{U}_i^X \beta$ is fulfilled using (at most) actions taken from X. We set $\Diamond_i^X \alpha \stackrel{\Delta}{\Longleftrightarrow} \top \mathcal{U}_i^X \alpha$ and $\Box_i^X \alpha \stackrel{\Delta}{\Longleftrightarrow} \sim \Diamond_i^X \sim \alpha$.

- $\alpha(i) \stackrel{\Delta}{\Longleftrightarrow} \alpha \mathcal{U}_i \alpha$ (or equivalently $\bot \mathcal{U}_i \alpha$). $\alpha(i)$ is to be read as "α at i". If $M = (T, \{V_i\})$ is a model and $c \in \mathcal{C}_T$ then $M, c \models \alpha(i)$ iff $M, \downarrow^i(c) \models \alpha$. It could of course be the case that $\mathrm{loc}(\alpha) \neq \{i\}$.

A simple but important observation is that every formula is a boolean combination of formulas taken from $\bigcup_{i \in \mathcal{P}} \Phi^i$. In TrPTL we can say that a specific global configuration is reachable from the initial configuration. Let $\{\alpha_i\}_{i \in \mathcal{P}}$ be a family with $\alpha_i \in \Phi^i$ for each i. Then we can define a derived connective $\Diamond(\alpha_1, \alpha_2, \ldots, \alpha_K)$ which has the following semantics at the empty configuration. Let $M = (T, \{V_i\})$ be a model. Then $M, \emptyset \models \Diamond(\alpha_1, \alpha_2, \ldots, \alpha_k)$ iff there exists $c \in \mathcal{C}_T$ such that $M, c \models \alpha_1 \wedge \alpha_2 \wedge \cdots \wedge \alpha_K$.

To define this derived connective set $\Sigma_1' = \Sigma_1$ and, for $1 < i \leq K$, set $\Sigma_i' = \Sigma_i - \cup\{\Sigma_j \mid 1 \leq j < i\}$. Then $\Diamond(\alpha_1, \alpha_2, \ldots, \alpha_K)$ is the formula:

$$\Diamond_1^{\Sigma_1'}(\alpha_1 \wedge \Diamond_2^{\Sigma_2'}(\alpha_2 \wedge \Diamond_3^{\Sigma_3'}(\alpha_3 \wedge \cdots \Diamond_K^{\Sigma_K'} \alpha_K)) \cdots).$$

The idea is that the sequence of actions leading up to the required configuration can be reordered so that one first performs all the actions in Σ_1, then all the actions in $\Sigma_2 - \Sigma_1$ etc. Hence, if now is an atomic proposition, the formula $\Diamond(\mathsf{now}(1), \mathsf{now}(2), \ldots, \mathsf{now}(K))$ is satisfied at the empty configuration iff there is a reachable configuration at which all the agents assert now.

Dually, safety properties that hold at the initial configuration can also be expressed. For example, let crt_i be the atomic assertion declaring that the agent i is currently in its critical section. Then it is possible to write a formula φ_{ME} which asserts that at all reachable configurations at most one agent is in its critical section, thereby guaranteeing that the system satisfies the mutual exclusion property. We omit the details of how to specify φ_{ME}.

On the other hand, it seems difficult to express nested global and safety properties in TrPTL. It is also the case that due to the local nature of the modalities, information about the past sneaks into the semantics even though there are no explicit past operators in the logic.

A formula α is said to be *root-satisfiable* iff there exists a model M such that $M, \emptyset \models \alpha$. On the other hand, α is said to be *satisfiable* iff there exists a model $M = (T, \{V_i\})$ and $c \in \mathcal{C}_T$ such that $M, c \models \alpha$. It turns out that these two notions are *not* equivalent. Consider the distributed alphabet $\widetilde{\Sigma}_0 = \{\Sigma_1, \Sigma_2\}$ with $\Sigma_1 = \{a, d\}$ and $\Sigma_2 = \{b, d\}$. Then it is not difficult

to verify that the formula $p(2)(1) \wedge \Box_2 \sim p(2)$ is satisfiable but not root-satisfiable. (Recall that $p(2)(1)$ abbreviates $\bot \mathcal{U}_1 p(2)$). One can however transform every formula α into a formula α' such that α is satisfiable iff α' is root satisfiable.

This follows from the observation that every α can be expressed as a boolean combination of formulas taken from the set $\bigcup_{i \in \mathcal{P}} \Phi^i$. Hence the given formula α can be assumed to be of the form $\alpha = \bigvee_{j=1}^{m} (\alpha_{j1} \wedge \alpha_{j2} \wedge \cdots \wedge \alpha_{jK})$ where $\alpha_{ji} \in \Phi^i$ for each $j \in \{1, 2, \ldots, m\}$ and each $i \in \mathcal{P}$. Now convert α to the formula α' where $\alpha' = \bigvee_{j=1}^{m} \Diamond(\alpha_{j1}, \alpha_{j2}, \cdots, \alpha_{jK})$. (Recall the derived modality $\Diamond(\alpha_1, \alpha_2, \ldots, \alpha_K)$ introduced earlier.) From the semantics of $\Diamond(\alpha_1, \alpha_2, \ldots, \alpha_K)$ it follows that α is satisfiable iff α' is root-satisfiable.

Hence, in principle, it suffices to consider only root-satisfiability in developing a decision procedure for TrPTL. There is of course a blow-up involved in converting satisfiable formulas to root-satisfiable formulas. If one wants to avoid this blow-up then the decision procedure for checking root-satisfiability can be suitably modified to yield a direct decision procedure for checking satisfiability as done in [44]. In any case, it is root satisfiability which is of importance from the standpoint of model checking. Hence here we shall only develop a procedure for deciding if a given formula of TrPTL is root-satisfiable.

As a first step we augment the syntax of our logic by one more construct.

- If α is a formula, so is $O_i\alpha$. In the model $M = (T, \{V_i\})$, at the configuration $c \in \mathcal{C}_T$, $M, c \models O_i\alpha$ iff $M, c \models \langle a \rangle_i \alpha$ for some $a \in \Sigma_i$. We also define $\mathrm{loc}(O_i\alpha) = \{i\}$.

Secondly, we will from now on drop the atomic propositions and instead work with the constant $\top$ and its negation $\bot$ as done earlier. The semantic definitions are assumed to be suitably modified.

Thus $O_i\alpha \equiv \bigvee_{a \in \Sigma_i} \langle a \rangle_i \alpha$ is a valid formula and O_i is expressible in the former syntax. It will be however more efficient to admit O_i as a first class modality as we did in Section 2.

Fix a formula α_0. Our aim is to effectively associate an asynchronous automaton $\mathcal{A}_{\alpha_0}$ with α_0 such that α_0 is root-satisfiable iff $L_{Tr}(\mathcal{A}_{\alpha_0}) \neq \emptyset$. Since the emptiness problem for asynchronous automata is decidable (Proposition 6.3), this will yield the desired decision procedure. Let $cl(\alpha_0)$ be the least set of formulas containing α_0 which satisfies:

- $\sim\alpha \in cl(\alpha_0)$ implies $\alpha \in cl(\alpha_0)$.
- $\alpha \vee \beta \in cl(\alpha_0)$ implies $\alpha, \beta \in cl(\alpha_0)$.

- $\langle a \rangle_i \alpha \in cl(\alpha_0)$ implies $\alpha \in cl(\alpha_0)$.
- $O_i \alpha \in cl(\alpha_0)$ implies $\alpha \in cl(\alpha_0)$.
- $\alpha \mathcal{U}_i \beta \in cl(\alpha_0)$ implies $\alpha, \beta \in cl(\alpha_0)$. In addition, $O_i(\alpha \mathcal{U}_i \beta) \in cl(\alpha_0)$.

We then define $CL(\alpha_0)$ to be the set $cl(\alpha_0) \cup \{\sim\beta \mid \beta \in cl(\alpha_0)\}$.

Thus $CL(\alpha_0)$, sometimes called the Fisher-Ladner closure of α_0, is closed under negation with the convention that $\sim\sim\beta$ is identified with β. Moreover, throughout the remainder of the section all formulas that we encounter will be assumed to be members of $CL(\alpha_0)$. From now we shall write CL instead of $CL(\alpha_0)$.

$A \subseteq CL$ is called an *i-type atom* iff it satisfies:

- $\top \in A$.
- $\alpha \in A$ iff $\sim\alpha \notin A$.
- $\alpha \vee \beta \in A$ iff $\alpha \in A$ or $\beta \in A$.
- $\alpha \mathcal{U}_i \beta \in A$ iff $\beta \in A$ or ($\alpha \in A$ and $O_i(\alpha \mathcal{U}_i \beta) \in A$).
- If $\langle a \rangle_i \alpha$, $\langle b \rangle_i \beta \in A_i$ then $a = b$.

AT_i denotes the set of i-type atoms. We now need to define the notion of a formula in CL being a member of a collection of atoms. Let $\alpha \in CL$ and $\{A_i\}_{i \in Q}$ be a family of atoms with $\mathrm{loc}(\alpha) \subseteq Q$ and $A_i \in AT_i$ for each $i \in Q$. We'll define the predicate Member(α,$\{A_i\}_{i \in Q}$), which for convenience will be denoted by $\alpha \in \{A_i\}_{i \in Q}$. It is defined inductively as:

- If $\mathrm{loc}(\alpha) = \{j\}$ then $\alpha \in \{A_i\}_{i \in Q}$ iff $\alpha \in A_j$.
- If $\alpha = \sim\beta$ then $\alpha \in \{A_i\}_{i \in Q}$ iff $\beta \notin \{A_i\}_{i \in Q}$.
- If $\alpha = \alpha_1 \vee \alpha_2$ then $\alpha_1 \vee \alpha_2 \in \{A_i\}_{i \in Q}$ iff $\alpha_1 \in \{A_i\}_{i \in Q}$ or $\alpha_2 \in \{A_i\}_{i \in Q}$.

The construction of the asynchronous automaton $\mathcal{A}_{\alpha_0}$ is guided by the construction developed for LTL in Section 2. However in the much richer setting of traces it turns out that one must make crucial use of the latest information that the agents have about each other when defining the transitions of $\mathcal{A}_{\alpha_0}$. It has been shown by Mukund and Sohoni [29] that this information can be kept track of by a deterministic asynchronous automaton whose size depends only on $\widetilde{\Sigma}$. (Actually the automaton described in [29] operates over finite traces but it is a trivial task to convert it into an asynchronous automaton having the desired properties). To bring out the

relevant properties of this automaton, let $T \in TR^\omega$ with $T = (E, \leq, \lambda)$. For each subset Q of processes, the function $\mathsf{latest}_{T,Q} : \mathcal{C}_T \times \mathcal{P} \to Q$ is given by $\mathsf{latest}_{T,Q}(c, j) = \ell$ iff ℓ is the least member of Q (under the usual ordering over the integers) with the property $\downarrow^j(\downarrow^q(c)) \subseteq \downarrow^j(\downarrow^\ell(c))$ for every $q \in Q$. In other words, among the agents in Q, ℓ has the best information about j at c, with ties being broken by the usual ordering over integers.

Theorem 7.3 ([29]) *There exists an effectively constructible deterministic asynchronous automaton $\mathcal{A}_\Gamma = (\{\Gamma_i\}, \{\Longrightarrow_a\}, \Gamma_{in}, \{(F_i, F_i^\omega)\})$ such that:*

1. *$L_{Tr}(\mathcal{A}_\Gamma) = TR^\omega$.*
2. *For each $Q = \{i_1, i_2, \ldots, i_n\}$, there exists an effectively computable function $\mathsf{gossip}_Q : \Gamma_{i_1} \times \Gamma_{i_2} \times \cdots \times \Gamma_{i_n} \times \mathcal{P} \to Q$ such that for every $T \in TR^\omega$, every $c \in \mathcal{C}_T$ and every $j \in \mathcal{P}$, $\mathsf{latest}_{T,Q}(c, j) = \mathsf{gossip}_Q(\gamma(i_1), \ldots, \gamma(i_n), j)$ where $\rho_{_T}(c) = \gamma$ and $\rho_{_T}$ is the unique (accepting) run of $\mathcal{A}_\Gamma$ over T.*

Henceforth, we refer to $\mathcal{A}_\Gamma$ as the *gossip automaton*. Each process in the gossip automaton has $2^{O(K^2 \log K)}$ local states, where $K = |\mathcal{P}|$. Moreover the function gossip_Q can be computed in time which is polynomial in the size of K.

Each i-state of the automaton $\mathcal{A}_{\alpha_0}$ will consist of an i-type atom together with an appropriate i-state of the gossip automaton. Two additional components will be used to check for liveness requirements. One component will take values from the set $N_i = \{0, 1, 2, \ldots, |U_i|\}$ where $U_i = \{\alpha \mathcal{U}_i \beta \mid \alpha \mathcal{U}_i \beta \in CL\}$. This component will be used to ensure that all "until" requirements are met. The other component will take values from the set $\{\mathsf{on}, \mathsf{off}\}$. This will be used to detect when an agent has quit.

The automaton $\mathcal{A}_{\alpha_0}$ can now be defined as:

$$\mathcal{A}_{\alpha_0} = (\{S_i\}, \{\longrightarrow_a\}, S_{in}, \{(F_i, F_i^\omega)\}),$$

where:

- For each i, $S_i = AT_i \times \Gamma_i \times N_i \times \{\mathsf{on}, \mathsf{off}\}$. Recall that Γ_i is the set of i-states of the gossip automaton and $N_i = \{0, 1, 2, \ldots, |U_i|\}$ with $U_i = \{\alpha \mathcal{U}_i \beta \mid \alpha \mathcal{U}_i \beta \in CL\}$.
- Let $s_a, s'_a \in S_a$ with $s_a(i) = (A_i, \gamma_i, n_i, v_i)$ and $s'_a(i) = (A'_i, \gamma'_i, n'_i, v'_i)$ for each $i \in \mathrm{loc}(a)$. Then $(s_a, s'_a) \in \longrightarrow_a$ iff the following conditions are met.

- $(\gamma_a, \gamma'_a) \in \Longrightarrow_a$ (recall that $\{\Longrightarrow_a\}$ is the family of transition relations of the gossip automaton) where $\gamma_a, \gamma'_a \in \Gamma_a$ such that $\gamma_a(i) = \gamma_i$ and $\gamma'_a(i) = \gamma'_i$ for each $i \in \mathrm{loc}(a)$.
- $\forall i, j \in \mathrm{loc}(a)$, $A'_i = A'_j$.
- $\forall i \in \mathrm{loc}(a)\ \forall \langle a \rangle_i \alpha \in CL$. $\langle a \rangle_i \alpha \in A_i$ iff $\alpha \in A'_i$.
- $\forall i \in \mathrm{loc}(a)\ \forall O_i \alpha \in CL$. $O_i \alpha \in A$ iff $\alpha \in A'_i$.
- $\forall i \in \mathrm{loc}(a) \forall \langle b \rangle_i \beta \in CL$. If $\langle b \rangle_i \beta \in A_i$ then $b = a$.
- Suppose $j \notin \mathrm{loc}(a)$ and $\beta \in CL$ with $\mathrm{loc}(\beta) = \{j\}$. Further suppose that $\mathrm{loc}(a) = \{i_1, i_2, \ldots, i_n\}$. Then $\beta \in A'_i$ iff $\beta \in A_\ell$ where $\ell = \mathsf{gossip}_{\mathrm{loc}(a)}(\gamma_{i_1}, \gamma_{i_2}, \ldots, \gamma_{i_n}, j)$.
- Let $i \in \mathrm{loc}(a)$, $U_i = \{\alpha_1 \mathcal{U}_i \beta_1, \alpha_2 \mathcal{U}_i \beta_2, \ldots, \alpha_{n_i} \mathcal{U}_i \beta_{n_i}\}$. Then u'_i and u_i are related to each other via:

$$u'_i = \begin{cases} (u_i{+}1) \bmod (n_i{+}1), & \text{if } u_i = 0 \text{ or } \beta_{u_i} \in A_i \text{ or } \alpha_{u_i} \mathcal{U}_i \beta_{u_i} \notin A_i \\ u_i, & \text{otherwise} \end{cases}$$

- For each $i \in \mathrm{loc}(a)$, $v_i = \mathsf{on}$. Moreover, if $v'_i = \mathsf{off}$ then $\langle a \rangle_i \alpha \notin A'_i$ for every $i \in \mathrm{loc}(a)$ and every $\langle a \rangle_i \alpha \in CL$.

- Let $s \in S_{\mathcal{P}}$ with $s(i) = (A_i, \gamma_i, u_i, v_i)$ for every i. Then $s \in S_{in}$ iff $\alpha_0 \in \{A_i\}_{i \in \mathcal{P}}$ and $\gamma \in \Gamma_{in}$ where $\gamma \in \Gamma_{\mathcal{P}}$ satisfies $\gamma(i) = \gamma_i$ for every i. Furthermore, $u_i = 0$ for every i. Finally, for every i, $v_i = \mathsf{off}$ implies that $\langle a \rangle_i \alpha \notin A_i$ for every $\langle a \rangle_i \alpha \in CL$.

- For each i, $F_i^\omega \subseteq S_i$ is given by $F_i^\omega = \{(A_i, \gamma_i, u_i, v_i) \mid u_i = 0 \text{ and } v_i = \mathsf{on}\}$ and $F_i \subseteq S_i$ is given by $F_i = \{(A_i, \gamma_i, u_i, v_i) \mid v_i = \mathsf{off}\}$.

This construction is an optimized version of the original construction for TrPTL presented in [44, 45]. Note that $\mathcal{A}_{\alpha_0}$ is indeed in standard form. Arguments similar to those presented in [44, 45] lead to the next set of results.

Theorem 7.4

1. *α_0 is root-satisfiable iff $L_{Tr}(\mathcal{A}_{\alpha_0}) \neq \emptyset$.*
2. *The number of local states of $\mathcal{A}_{\alpha_0}$ is bounded by $2^{O(\max(n, m^2 \log m))}$ where $n = |\alpha_0|$ and m is the number of agents mentioned in α_0. Clearly, $m \leq n$. It follows that the root-satisfiability problem (and in fact the satisfiability problem) for* TrPTL *is solvable in time $2^{O(\max(n, m^2 \log m) \cdot m)}$.*

The number of local states of each process in $\mathcal{A}_{\alpha_0}$ is determined by two quantities: the length of α_0 and the size of the gossip automaton $\mathcal{A}_\Gamma$. As far as the size of $\mathcal{A}_\Gamma$ is concerned, it is easy to verify that we need to consider only those agents in $\mathcal{P}$ that are mentioned in $\mathrm{loc}(\alpha_0)$, rather than all agents in the system.

The *model checking problem* for TrPTL can be phrased as follows. A finite state distributed program Pr over $\widetilde{\Sigma}$ is an asynchronous automaton $\mathcal{A}_{Pr} = (\{S_i^{Pr}\}, \{\Longrightarrow_a^{Pr}\}, S_{in}^{Pr}, \{(S_i^{Pr}, S_i^{Pr})\})$ modelling the state space of Pr.

Viewing a formula α_0 as a specification, we say that Pr *meets the specification* α_0 — denoted $Pr \models \alpha_0$ — if for every $T \in TR^\omega$, if $\mathcal{A}_{Pr}$ has a run over T then $T, \emptyset \models \alpha_0$.

The model checking problem for TrPTL can be solved by "intersecting" the program automaton $\mathcal{A}_{Pr}$ with the formula automaton $\mathcal{A}_{\sim\alpha_0}$ to yield an automaton $\mathcal{A}$ such that $L_{Tr}(\mathcal{A}) = L_{Tr}(\mathcal{A}_{Pr}) \cap L_{Tr}(\mathcal{A}_{\sim\alpha_0})$. As before, $L_{Tr}(\mathcal{A}) = \emptyset$ iff $Pr \models \alpha_0$.

It turns out that this model checking problem has time complexity $O(|\mathcal{A}_{Pr}| \cdot 2^{O(\max(n, m^2 \log m) \cdot m)})$ where $|\mathcal{A}_{Pr}|$ is the size of the global state space of the asynchronous automaton modelling the behaviour of the given program Pr and, as before, $n = |\alpha_0|$ and m is the number of agents mentioned in α_0, where α_0 is the specification formula.

We now take a brief look at some related agent-based linear time temporal logics over traces. The first one is the sublogic of TrPTL denoted which consists of the so called connected formulas of TrPTL. We define $\Phi^{\mathrm{con}}_{\mathrm{TrPTL}}$ (from now on written as Φ^{con}) to be the least subset of Φ satisfying the following conditions:

- $\top \in \Phi^{\mathrm{con}}$ and as before $\mathrm{Loc}(\top) = \emptyset$
- If $\alpha, \beta \in \Phi^{\mathrm{con}}$, so are $\sim\alpha$ and $\alpha \vee \beta$.
- If $\alpha \in \Phi^{\mathrm{con}}$ and $a \in \Sigma_i$ such that $\mathrm{loc}(\alpha) \subseteq \mathrm{loc}(a)$ then $\langle a \rangle_i \alpha \in \Phi^{\mathrm{con}}$.
- If $\alpha, \beta \in \Phi^{\mathrm{con}}$ with $\mathrm{loc}(\alpha) = \mathrm{loc}(\beta) = \{i\}$ then $\alpha \mathcal{U}_i \beta \in \Phi^{\mathrm{con}}$. Actually one need only demand that $\mathrm{loc}(\alpha)$, $\mathrm{loc}(\beta) \subseteq \bigcap\{\mathrm{loc}(a) \mid a \in \Sigma_i\}$ but this leads to notational complications that we wish to avoid here.
- If $\alpha \in \Phi^{\mathrm{con}}$ and $\mathrm{loc}(\alpha) = \{i\}$ then $O_i \alpha \in \Phi^{\mathrm{con}}$. (Once again one needs to just demand that $\alpha \subseteq \bigcap\{\mathrm{loc}(a) \mid a \in \Sigma_i\}$.)

Connected formulas were first identified by Niebert and used by Huhn [22]. They have also been independently identified by Ramanujam [38]. Thanks to the syntactic restrictions imposed on the next state and until formulas, past information is not allowed to creep in. Indeed one can prove the following:

Proposition 7.5 *Let $\alpha \in \Phi^{\mathrm{con}}$. Then α is satisfiable iff α is root-satisfiable.*

Yet another pleasing feature of TrPTL$^{\mathrm{con}}$ is that the gossip automaton can be eliminated in the construction of the automaton $\mathcal{A}_{\alpha_0}$ whenever $\alpha_0 \in \Phi^{\mathrm{con}}$. In fact one can prove the following.

Theorem 7.6 *The satisfiability problem for* TrPTL$^{\mathrm{con}}$ *is solvable in time* $2^{O(|\alpha_0|)}$.

Once again, a suitably modified statement can be made about the associated model checking problem. At present we do not know whether or not TrPTL is strictly more expressive than TrPTL$^{\mathrm{con}}$, but it is clear that LTL$^{\otimes}$ is a strict sublogic of TrPTL$^{\mathrm{con}}$. We shall deal with the relative strengths of these logics in the next section. Two of the four logics considered by Ramanujam [38] in a closely related setting turn out to be LTL$^{\otimes}$ and TrPTL$^{\mathrm{con}}$. We conjecture that the other two logics are also expressible within TrPTL.

Katz and Peled introduced the logic ISTL [24] whose semantics has a trace-theoretic flavour. In a subsequent paper by Peled and Pnueli [34] on ISTL, the connection to traces was made more directly. Indeed this is one of the first instances of the explicit use of traces in a temporal logical setting that we know of. However, it has branching time modalities which permit quantification over the so called observations of a trace. ISTL uses global atomic propositions rather than local atomic propositions. Penczek has also studied a number of temporal logics (including a version of ISTL) with branching time modalities and global atomic propositions [36]. His logics are interpreted directly over the space of configurations of a trace resulting in a variety of axiomatizations and undecidability results. We feel that local atomic propositions (as used in TrPTL) are crucial for obtaining tractable partial order based temporal logics. Niebert has considered several μ-calculus versions of TrPTL [31, 32] and has obtained various decidability results using a variant of asynchronous Büchi automata.

The temporal logic of causality (TLC) proposed by Alur, Peled and Penczek is basically a temporal logic over traces [1]. The concurrent structures used in [1] as frames for TLC can be easily represented as traces over an appropriately chosen trace alphabet. The interesting feature of TLC is that its branching time modalities are interpreted over causal paths. In a trace $(E, \leq, \lambda)$, the sequence $e_0 e_1 \cdots \in E^{\infty}$ is a causal path if $e_0 \lessdot e_1 \lessdot e_2 \cdots$. This logic admits an essentially exponential time decision procedure for checking satisfiablity in terms of a variant of Büchi automata called Street automata.

8 Expressiveness Issues

Our aim here is to discuss some expressiveness issues concerning trace-based linear time temporal logics. To set the stage we first quickly review the classical case of sequences.

The *monadic second-order theory of infinite sequences over* Σ is denoted MSO(Σ). Its vocabulary consists of a family of unary predicates $\{R_a\}_{a\in\Sigma}$, one for each $a \in \Sigma$; a binary predicate $\leq$; a binary predicate $\in$; a countable supply of individual variables $Var = \{x, y, z, \ldots\}$; a countable supply of set variables (i.e. monadic predicate variables) $SVar = \{X, Y, Z, \ldots\}$. The formulas of MSO($\Sigma$) are then built up by:

- $R_a(x)$, $x \leq y$ and $x \in X$ are atomic formulas.
- If ϕ and ϕ' are formulas then so are $\sim\phi$, $\phi \vee \phi'$, $(\exists x)\phi$ and $(\exists X)\phi$.

A structure for MSO(Σ) is a ω-sequence $\sigma \in \Sigma^\omega$. Let $\mathcal{I}$ be an interpretation of the variables with $\mathcal{I} : Var \longrightarrow \omega$ and $\mathcal{I} : SVar \longrightarrow 2^\omega$. Then the notion of σ being a model of ϕ under the interpretation $\mathcal{I}$, denoted $\sigma \models_{\mathcal{I}} \phi$, is defined in the expected manner. In particular, $\sigma \models_{\mathcal{I}} R_a(x)$ iff $\sigma(\mathcal{I}(x)) = a$ (note that $\sigma \in \Sigma^\omega$ is viewed as $\sigma : \omega \longrightarrow \Sigma$); $\sigma \models_{\mathcal{I}} x \leq y$ iff $\mathcal{I}(x) \leq \mathcal{I}(y)$ (here $\leq$ is the usual ordering over ω); $\sigma \models_{\mathcal{I}} x \in X$ iff $\mathcal{I}(x) \in \mathcal{I}(X)$.

As usual, a sentence is a formula with no free variables. Each sentence ϕ defines an ω-language, denoted L_ϕ, where:

$$L_\phi = \{\sigma \mid \sigma \models \phi\}.$$

We say that $L \subseteq \Sigma^\omega$ is MSO(Σ)-*definable* iff there exists a sentence $\phi \in$ MSO(Σ) such that $L = L_\phi$. A celebrated result of Büchi [4] shows that the class of languages expressible by sentences in MSO(Σ) coincides with the class of languages recognized by Büchi automata over Σ. This class is the *ω-regular languages* over Σ.

The *first-order theory of infinite sequences over* Σ is denoted FO(Σ) and is obtained from MSO(Σ) by abolishing the monadic second-order quantifications from the logic. The semantics and notions of first-order definability are carried over in the obvious manner.

A fundamental result in the theory of temporal logic is Kamp's Theorem [23] which was later strengthened in [14] to establish that LTL(Σ) is expressively equivalent to the FO(Σ). The surprise here being that LTL(Σ) admits only a bounded number of operators (one unary and one binary as we have formulated it) whereas infinitely many operators of increasing arities can be defined in FO(Σ). Secondly, as we saw in Section 2, the satisfiability

problem for LTL(Σ) can be solved in deterministic exponential time. The satisfiability problem for FO(Σ) on the other hand, even when the sentences are interpreted over *finite* words, is known to be non-elementary hard [43]. It is quite easy to see that FO(Σ) — and hence LTL(Σ) — is strictly less expressive than MSO(Σ) in the sense that there is a language which is MSO(Σ)-definable but not FO(Σ)-definable. (Indeed this is the sense in which we shall compare the expressive power of various logics in what follows.) For instance, as pointed out by Wolper in a state-based setting [56], the language $L \subseteq \{a, b\}^\omega$ given by "a is executed at every even position" is not definable in this logic. On the other hand, it is easy to come up with a formula of MSO(Σ) defining L.

The expressive power of LTL can be extended to obtain the expressive power of MSO while still guaranteeing an exponential time decidable satisfiability problem as demonstrated first in [57]. Here we sketch how the regular programs over Σ can be used to achieve this goal [19].

The syntax of regular programs over Σ 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.

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

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,

- $\sigma, \tau \models \alpha\, U^\pi \beta$ 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 $\alpha U^{\pi} \beta$, one must satisfy $\alpha U \beta$ along some finite stretch of behaviour which is required to be in the (linear time) behaviour of the program π. We associate with a formula α of DLTL(Σ) the ω-language L_α in the obvious manner.

A useful derived operator of DLTL is:

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

By replacing the until-modality of DLTL with the above derived operator we obtain the sublogic DLTL^{-}(Σ), which is essentially Propositional Dynamic Logic [13] equipped with a linear time semantics. It turns out that DLTL(Σ) and DLTL^{-}(Σ) both have the same expressive power as MSO(Σ).

Theorem 8.1 *Let $L \subseteq \Sigma^{\omega}$. Then the following statements are equivalent.*

1. *L is ω-regular (i.e. definable in* MSO(Σ)*).*
2. *L is* DLTL(Σ)*-definable.*
3. *L is* DLTL^{-}(Σ)*-definable.*

Both the satisfiablity and model checking problems for DLTL(Σ) are decidable with the same time complexity as for LTL(Σ).

Let (Σ, I) be trace alphabet. Then MSO(Σ, I), the *monadic second-order theory of infinite traces (over Σ, I)*, has the same syntax as MSO(Σ). The structures are elements of $TR^{\omega}(\Sigma, I)$. Let $T \in TR^{\omega}(\Sigma, I)$ with $T = (E, \leq, \lambda)$ and let $\mathcal{I} : X \to E$ be an interpretation. Then $T \models^{\mathrm{MSO}}_{\mathcal{I}} R_a(x)$ iff $\lambda(\mathcal{I}(x)) = a$ and $T \models^{\mathrm{MSO}}_{\mathcal{I}} x \leq y$ iff $\mathcal{I}(x) \leq \mathcal{I}(y)$. Hence, the essential difference is that the binary predicate symbols is now interpreted as the causal partial order of the trace. The remaining semantic definitions go along the expected lines. Each sentence φ (i.e., a formula with no free occurrences of variables) defines the ω-trace language

$$L_{\varphi} = \{T \mid T \models^{\mathrm{MSO}} \varphi\}.$$

We say that $L \subseteq TR^{\omega}$ is MSO-definable iff there exists a sentence φ in MSO(Σ, I) such that $L = L_{\varphi}$. It is known that MSO-definable languages are precisely the regular trace languages; i.e. those recognized by asynchronous automata [11].

FO(Σ, I), *the first-order theory of traces*, is defined in the obvious way. Clearly it will be strictly weaker than MSO(Σ, I). For more information the reader is referred to [7]. Naturally both these theories can be made to handle finite traces as well.

Through the rest of this section we fix a distributed alphabet $\widetilde{\Sigma}$ and let (Σ, I) be the induced trace alphabet. By $\mathrm{MSO}(\widetilde{\Sigma})$ we shall mean the theory $\mathrm{MSO}(\Sigma, I)$ and similarly for $\mathrm{FO}(\widetilde{\Sigma})$, the first-order fragment of $\mathrm{MSO}(\widetilde{\Sigma})$. In what follows we shall often supress the mention of $\widetilde{\Sigma}$ as well as the induced (Σ, I).

We first consider the logic $\mathrm{LTL}^{\otimes}$. Recall that product languages are trace consistent and hence they induce trace languages via the map str. The resulting trace languages will be called product trace languages. As might be expected, the regular product trace languages are the ones obtained from regular product languages via the map str. It is easy to show that not every (regular) trace language is a product trace language [47]. It is also easy to see that $\mathrm{LTL}^{\otimes}$-definable trace languages constitute a strict subclass of regular product trace languages. It has been shown that a product version of DLTL denoted $\mathrm{DLTL}^{\otimes}$ captures exactly the class of regular product trace languages [20]. We also claim that it is an easy exercise to formulate a product version of $\mathrm{MSO}(\widetilde{\Sigma})$ and show that it captures exactly the regular product trace languages. Let us denote this product version of $\mathrm{MSO}(\widetilde{\Sigma})$ as $\mathrm{MSO}^{\otimes}(\widetilde{\Sigma})$ and its first-order fragment as $\mathrm{FO}^{\otimes}(\widetilde{\Sigma})$. It is easy to show — using Kamp's theorem — that $\mathrm{LTL}^{\otimes}(\widetilde{\Sigma})$ has exactly the same expressive power as $\mathrm{FO}^{\otimes}(\widetilde{\Sigma})$.

We also know that $\mathrm{LTL}^{\otimes}$ is strictly weaker than TrPTL. First note that each formula (say α of TrPTL) defines a trace language L_{α} via :

$$L_{\alpha} = \{\, T \mid T, \emptyset \models \alpha \}.$$

Hence we can compare the relative expressive powers of $\mathrm{LTL}^{\otimes}$ and TrPTL. It is known that ([30, 47]):

$$\mathrm{LTL}^{\otimes} \subset \mathrm{TrPTL}^{\mathrm{con}} \subseteq \mathrm{TrPTL}.$$

It is still open whether $\mathrm{TrPTL}^{\mathrm{con}}$ is equal to TrPTL in expressive power.

It is not difficult to show that TrPTL is no more expressive than the first-order theory of traces but it is not known whether the converse also holds. It would be nice to have a linear time temporal logic over traces patterned after LTL which has the same expressive power as the first-order theory of traces. The motivation is provided by the next result [11]:

Proposition 8.2 *Let $L \subseteq \Sigma^{\omega}$. Then the following statements are equivalent.*

1. *L is trace consistent and $\mathrm{LTL}(\Sigma)$-definable.*
2. *$\{\mathsf{str}(\sigma) \mid \sigma \in L\}$ is $\mathrm{FO}(\Sigma, I)$-definable.*

Egged on by this result, recently a different kind of trace-based linear time temporal logic called LTrL has been proposed [48]. This logic works directly with a trace alphabet (i.e. it is not based on agents). It is interpreted over the configurations of a trace and its syntax is given by:

$$\mathrm{LTrL}(\Sigma, I) ::= \top \mid \sim\alpha \mid \alpha \vee \beta \mid \langle a\rangle\alpha \mid \alpha \; U \; \beta \mid \langle a^{-1}\rangle\top.$$

Thus the syntax is very close to LTL except for the addition of a very restricted past-operator. In fact, just a *constant* number of past-operators are present in the logic; one for each action.

A model of $\mathrm{LTrL}(\Sigma, I)$ is a trace $T = (E, \leq, \lambda)$. Let $c \in \mathcal{C}_T$ be a configuration of T. Then $T, c \models \alpha$ will stand for α being satisfied at c in T. This notion is defined inductively as follows:

- $T, c \models \top$.
- $T, c \models \sim\alpha$ and $T, c \models \alpha \vee \beta$ are defined in the expected manner.
- $T, c \models \langle a\rangle\alpha$ iff there exists $c' \in \mathcal{C}_T$ with $c \xrightarrow{a}_T c'$ with $T, c' \models \alpha$.
- $T, c \models \alpha \; U \; \beta$ iff there exists $c' \in \mathcal{C}_T$ with $c \subseteq c'$ such that $T, c' \models \beta$. Moreover, for every $c'' \in \mathcal{C}_T$, $c \subseteq c'' \subset c'$ implies $T, c'' \models \alpha$.
- $T, c \models \langle a^{-1}\rangle\alpha$ iff there exists $c' \in \mathcal{C}_T$ with $c' \xrightarrow{a}_T c$.

The major result concerning LTrL is the following:

Theorem 8.3 ([48]) *Let $L \subseteq TR^{\omega}(\Sigma, I)$. Then the following statements are equivalent.*

1. *L is $\mathrm{FO}(\Sigma, I)$-definable.*
2. *L is $\mathrm{LTrL}(\Sigma, I)$-definable.*

Thus — except for the addition of the restricted past-operators — LTrL is a generalization of Kamp's Theorem to the much richer setting of traces. Meyer and Petit have shown that the past-operators can be eliminated without loss of expressive power when the logic is interpreted over *finite* traces [28]. A similar result for infinite traces is not known at present. Unfortunately this logic does not have a matching time complexity in relation to LTL. Recently Walukiewicz has shown that the satisfiability problem for LTrL is non-elementary hard [53]. A related result concerns the logic TLPO formulated by Ebinger [10]. This is also a linear time temporal logic interpreted over traces but with full-fledged past-operators. TLPO is claimed to be expressively complete when interpreted over *finite* traces but nothing is known

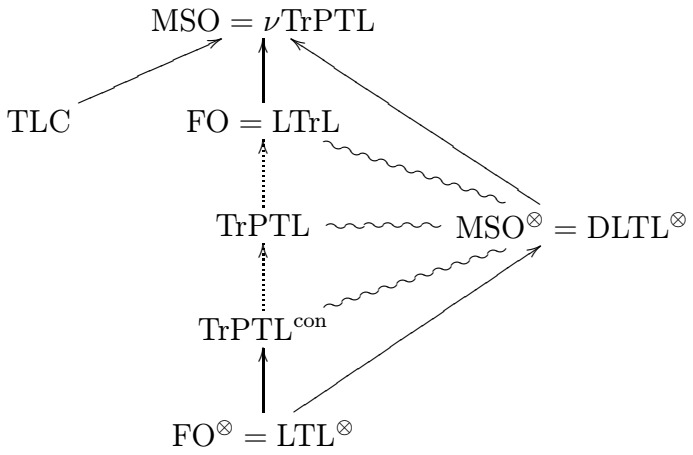

Figure 4: Relative expressive power of the logics

about the complexity of the satisfiability problem nor about its expressive power in relation to infinite traces.

At present we do not know much about the relationship between TLC and the logics we have mentioned so far, except that it is strictly weaker than the monadic second-order theory of traces.

In an interesting recent development Niebert [32] has formulated a fixed point based linear time temporal logic for traces in the setting of distributed alphabets. This logic is denoted as νTrPTL. It is equal in expressive power to the monadic second-order theory of traces *and* it has decision procedure of essentially exponential time complexity. However, the formulas of this logic are required to satisfy what appears to be awkward syntactic restrictions and it is not clear how one could express global properties of interest in this formalism.

The relative strengths of the various linear time temporal logics over traces mentioned in this section are displayed in Figure 4. A dotted (solid) arrow from A to B indicates that B is at least as expressive as (strictly more expressive than) A. Squiggled lines denote that the logics are incomparable to each other.

To conclude this section, a lot is known about linear time temporal logics for traces but at present we still do not have — unlike the case of sequences — pleasing counterparts to the first-order and monadic second-order theories of traces.

9 Conclusion

In this paper we have attempted an overview of linear time temporal logics interpreted over traces. We have mainly concentrated on the satisfiability and model checking problems as well as expressiveness issues. The problem of axiomatizing these logics seems to be a non-trivial task. Some partial results may be found in [39]. In [34] the authors present proof rules for the logic ISTL with a trace semantics together with a relative expressive completeness result. Reisig has also developed a kit of proof rules for a version of UNITY logic [40, 41]. The models of this logic are the non-sequential processes of a net system and the proof rules are mainly designed to help reason about distributed algorithms modelled using net systems.

At present not much is known about corresponding logics in a branching time setting. Most of the attempts in this direction have lead to logics whose satisfiablity problems are undecidable [5, 25, 36]. It is however the case that the model checking problem often remains tractable [5, 36]. We do not know at present whether the properties expressible in such logics have any type of "all-or-none" flavour and if so whether one can develop some reduction techniques for verifying such properties. Some preliminary attempts in this direction have been made in [16, 54].

References

[1] Alur, R., Peled, D., Penczek, W.: Model checking of causality properties. Proceedings of the 10th Annual IEEE Symposium on Logic in Computer Science, IEEE Computer Society Press (1995) 90–100

[2] Bell Labs Design Automation: FormalCheck$^{\mathrm{tm}}$. Further information can be obtained at `http://www.bell-labs.com/formalcheck/`

[3] Bracho, F., Droste, M., Kuske, D.: Representation of computations in concurrent automata by dependence orders. Theoretical Computer Science **174**(1-2) (1997) 67–96

[4] Büchi, J. R.: On a decision method in restricted second order arithmetic. Proceedings of the International Congress on Logic, Methodology and Philosophy of Science, Stanford University Press (1960) 1–11

[5] Cheng, A.: Petri nets, traces, and local model checking. Proceedings of the 4th International Conference on Algebraic Methodology and Software Technology, Lecture Notes in Computer Science **936**, Springer-Verlag (1995) 322–337

[6] Diekert, V.: Combinatorics of traces. Lecture Notes in Computer Science **454**, Springer-Verlag (1990)

[7] Diekert, V., Rozenberg, G. (eds.): The book of traces. World Scientific (1995)

[8] Droste, M.: Recognizable languages in concurrency monoids. Theoretical Computer Science **150**(1) (1995) 77–109

[9] Droste, M., Gastin, P.: Asynchronous cellular automata for pomsets without auto-concurrency. Proceedings of the 7th International Conference on Concurrency Theory, Lecture Notes in Computer Science **1119**, Springer-Verlag (1996) 627–638

[10] Ebinger, W.: Charakterisierung von sprachklassen unendlicher spuren durch logiken. Dissertation, Institut für Informatik, Universität Stuttgart (1994)

[11] Ebinger, W., Muscholl, A.: Logical definability on infinite traces. Theoretical Computer Science **154**(1) (1996) 67–84

[12] Emerson, A. E.: Temporal and modal logic. In Handbook of Theoretical Computer Science, volume B: Formal Models and Semantics, Elsevier Science Publishers (1990) 996–1072

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

[14] 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

[15] Gastin, P., Petit, A.: Asynchronous cellular automata for infinite traces. Proceedings of the 19th International Colloquium on Automata, Languages and Programming. Lecture Notes in Computer Science **623**, Springer-Verlag (1992) 583–594

[16] Gerth, R., Kuiper, R., Peled, D., Penczek, W.: A partial-order approach to branching time model checking. Proceedings of the 3rd Israeli Symposium on Theory of Computing and Systems, IEEE Computer Society Press (1995) 130–139

[17] Godefroid, P.: Partial-order methods for the verification of concurrent systems. Lecture Notes in Computer Science **1032**, Springer-Verlag (1996)

[18] Gerth, R., Peled, D., Vardi, M., Wolper, P.: Simple on-the-fly automatic verification of linear time temporal logic. Proceedings of the 15th IFIP WG 6.1 International Workshop on Protocol Specification, Testing, and Verification, North-Holland (1995)

[19] Henriksen, J. G., Thiagarajan, P. S.: Dynamic linear time temporal logic. Journal of Pure and Applied Logic, Elsevier (to appear)

[20] Henriksen, J. G., Thiagarajan, P. S.: A product version of dynamic linear time temporal logic. Proceedings of the 8th International Conference on Concurrency Theory, Lecture Notes in Computer Science **1243**, Springer-Verlag (1997) 45–58

[21] Holzmann, G. J.: An overview of the SPIN model checker. In "On-the-fly Model Checking Tutorial", BRICS Autumn School on Verification, Note NS-96-6, BRICS, Department of Computer Science, University of Aarhus (1996)

[22] Huhn, M.: On semantic and logical refinement of actions. Technical Report, Institut für Informatik, Universität Hildesheim, Germany (1996)

[23] Kamp, H. R.: Tense logic and the theory of linear order. Ph.D. thesis, University of California (1968)

[24] Katz, S., Peled, D.: Interleaving set temporal logic. Theoretical Computer Science **73**(3) (1992) 21–43

[25] Lodaya, K., Parikh, R., Ramanujam, R., Thiagarajan, P. S.: A logical study of distributed transition systems. Information and Computation **119**(1) (1995) 91–118

[26] Manna, Z., Pnueli, A.: The temporal logic of reactive and concurrent systems (specification), Springer-Verlag (1991)

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

[28] Meyer, R., Petit, A.: Expressive completeness of LTrL on finite traces: an algebraic proof. Proceedings of the 15th Annual Symposium on Theoretical Aspects of Computer Science **1373**, Lecture Notes in Computer Science, Springer-Verlag (1998) 533–543

[29] Mukund, M., Sohoni, M.: Keeping track of the latest gossip in a distributed system. Distributed Computing **10**(3) (1997) 117-127

[30] Mukund, M., Thiagarajan, P. S.: Linear time temporal logics over Mazurkiewicz traces. Proceedings of the 21st International Symposium on Mathematical Foundations of Computer Science, Lecture Notes in Computer Science **1113**, Springer-Verlag (1996) 62–92

[31] Niebert, P.: A ν-calculus with local views for systems of sequential agents. Proceedings of the 20th International Symposium on Mathematical Foundations of Computer Science, Lecture Notes in Computer Science **969**, Springer-Verlag (1995) 563–573

[32] Niebert, P.: A temporal logic for the specification and validation of distributed behaviour. Ph.D. thesis, University of Hildesheim (1997)

[33] Nielsen, M., Plotkin, G., Winskel, G.: Petri nets, event structures and domains, part I. Theoretical Computer Science **13**(1) (1981) 85–108

[34] Peled, D., Pnueli, A.: Proving partial order properties. Theoretical Computer Science **126**(2) (1994) 143–182

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

[36] Penczek, W.: Temporal logics for trace systems: on automated verification. International Journal of the Foundations of Computer Science **4**(1) (1993) 31–68

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

[38] Ramanujam, R.: Locally linear time temporal logic. Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science, IEEE Computer Society Press (1996) 118–127

[39] Ramanujam, R.: Rules for trace consistent reasoning. Proceedings of the 3rd Asian Computing Science Conference, Lecture Notes in Computer Science **1345**, Springer-Verlag (1997) 57–71

[40] Reisig, W.: Temporal logic and causality in concurrent systems. Proceedings of CONCURRENCY'88, Lecture Notes in Computer Science **335**, Springer-Verlag (1988) 121–139

[41] Reisig, W.: Petri net models for distributed algorithms. In Computer Science Today — Recent Trends and Developments, Lecture Notes in Computer Science **1000**, Springer-Verlag (1995) 441–454

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

[43] Stockmeyer, L. J.: The complexity of decision problems in automata theory and logic. Ph.D. thesis, MIT, Cambridge, Massachusetts (1974)

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

[45] Thiagarajan, P. S.: TrPTL: a trace based extension of linear time temporal logic. Technical report TCS-93-6, School of Mathematics, SPIC Science Foundation, Madras (1993)

[46] 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

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

[48] Thiagarajan, P. S., Walukiewicz, I.: An expressively complete linear time temporal logic for Mazurkiewicz traces. Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science, IEEE Computer Society Press (1997) 183–194

[49] Thomas, W.: Automata on infinite objects. In Handbook of Theoretical Computer Science, volume B: Formal Models and Semantics, Elsevier Science Publishers (1990) 133–191

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

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

[52] Vardi, M. Y.: An automata-theoretic approach to linear time temporal logic. In Logics for Concurrency - Structure vs. Automata, Lecture Notes in Computer Science **1043**, Springer-Verlag (1996) 238–266

[53] Walukiewicz, I.: Difficult configurations — on the complexity of LTrL (extended abstract). Proceedings of the 25th International Colloquium on Automata, Languages and Programming. Lecture Notes in Computer Science, Springer-Verlag (to appear)

[54] Willems, B., Wolper, P.: Partial-order methods for model checking: from linear time to branching time. Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science, IEEE Computer Society Press (1996) 294–303

[55] Winskel, G., Nielsen, M.: Models for concurrency. In Handbook of Logic and the Foundations of Computer Science, volume 4, Oxford University Press (1995) 1–148

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

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

[58] Zielonka, W.: Notes on finite asynchronous automata. R.A.I.R.O. Informatique Théorique et Applications **21** (1987) 99–135

Recent BRICS Report Series Publications

RS-98-8 **P. S. Thiagarajan and Jesper G. Henriksen.** ***Distributed Versions of Linear Time Temporal Logic: A Trace Perspective*****. April 1998. 49 pp. Appears as a chapter of Reisig and Rozenberg, editors,** ***Lectures on Petri Nets I: Basic Models*****, LNCS 1491, 1998, pages 643–681.**

RS-98-7 **Stephen Alstrup, Thore Husfeldt, and Theis Rauhe.** ***Marked Ancestor Problems (Preliminary Version)*****. April 1998. 36 pp.**

RS-98-6 **Kim Sunesen.** ***Further Results on Partial Order Equivalences on Infinite Systems*****. March 1998. 48 pp.**

RS-98-5 **Olivier Danvy.** ***Formatting Strings in ML*****. March 1998. 3 pp. This report is superseded by the later report BRICS RS-98-12.**

RS-98-4 **Mogens Nielsen and Thomas S. Hune.** ***Timed Bisimulation and Open Maps*****. February 1998. 27 pp. Appears in Brim, Gruska and Zlatuška, editors,** ***Mathematical Foundations of Computer Science: 23rd International Symposium*****, MFCS '98 Proceedings, LNCS 1450, 1998, pages 378–387.**

RS-98-3 **Christian N. S. Pedersen, Rune B. Lyngsø, and Jotun Hein.** ***Comparison of Coding DNA*****. January 1998. 20 pp. Appears in Farach-Colton, editor,** ***Combinatorial Pattern Matching: 9th Annual Symposium*****, CPM '98 Proceedings, LNCS 1448, 1998, pages 153–173.**

RS-98-2 **Olivier Danvy.** ***An Extensional Characterization of Lambda-Lifting and Lambda-Dropping*****. January 1998.**

RS-98-1 **Olivier Danvy.** ***A Simple Solution to Type Specialization (Extended Abstract)*****. January 1998. 7 pp. Appears in Larsen, Skyum and Winskel, editors,** ***25th International Colloquium on Automata, Languages, and Programming*****, ICALP '98 Proceedings, LNCS 1433, 1998, pages 908–917.**

RS-97-53 **Olivier Danvy.** ***Online Type-Directed Partial Evaluation*****. December 1997. 31 pp. Extended version of an article to appear in** ***Third Fuji International Symposium on Functional and Logic Programming*****, FLOPS '98 Proceedings (Kyoto, Japan, April 2–4, 1998), pages 271–295, World Scientific, 1998.**

RS-97-52 **Paola Quaglia.** ***On the Finitary Characterization of π-Congruences*****. December 1997. 59 pp.**