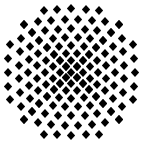


UNIVERSITÄT STUTTGART
FAKULTÄT INFORMATIK

Proceedings ASMICS Workshop
Infinite Traces

Tübingen, January 23 - 25, 1992

Volker Diekert, Werner Ebinger
(Editors)

Bericht 4/92

Preface

The workshop Infinite Traces was held at Tübingen, Hotel Hospiz, from January 23 to 25, 1992. The idea to this workshop was born at the general ASMICS meeting at Menaggio. It is the second ASMICS workshop on Free Parially Commutative Monoids, the first one was held at Kochel 1989. Whereas the Kochel workshop gave an representative overview over the different activities in the research field, the subject of the new workshop was restricted to the special theme of infinite traces with the aim of gathering only few experts.

The workshop was attended by 8 participants from six different European universities and in addition by three guests from the University of Stuttgart. The scientific programme started with lectures on Thursday, January 23 and ended at noon of Saturday January 25, 1992. In the afternoon of Saturday there was an excursion to the famous Hohenzollern castle at Hechingen about 30 km South of Tübingen.

The good working athmosphere at Hotel Hospiz and the active collaboration made this meeting to a successful event. We thank all participants for their contribution. We would like to express our gratitude to ASMICS and the Institut für Informatik of the University of Stuttgart for financial and other support. In paricular, we thank Heike Photien for local help. Last, but not least we thank Frau Elke Veihelmann from Hotel Hospiz for the warm and kind reception.

Stuttgart May 5, 1992

Volker Diekert Werner Ebinger

Programme
Workshop Infinite Traces
Tübingen, January 23-25, 1992, Hotel Hospiz

Thursday	January 23	
9.30 – 9.40	Volker Diekert	Opening
9.45 – 10.30	Giovanni Pighizzini	Operations on asynchronous automata
11.15 – 12.15	Paul Gastin	Büchi asynchronous cellular automata
16.15 – 17.15	Mike Stannett	Convergence in trace languages
Friday	January 24	
9.30 – 10.30	Hendrik Jan Hoogeboom	Automata on infinite words
11.00 – 12.00	Werner Ebinger	On logical definability of infinite trace languages
16.00 – 16.30	Mike Stannett	Transfinite traces
17.45 – 18.30	Wojciech Penczek	Temporal logic in trace systems
19.00 – 20.30	Volker Diekert (Chair)	Discussion
Saturday	January 25	
9.30 – 10.30	Antoine Petit	Poset properties of complex traces
10.30 – 11.00	Volker Diekert	Two extensions of existing models
11.15 – 12.45	Hendrik Jan Hoogeboom (Chair)	Open problems session
14.00 – 17.00	Excursion	

Contents

List of participants
Workshop Infinite Traces, Tbingen, January 23-25, 1992
Hotel Hospiz

Volker Diekert	Stuttgart
Werner Ebinger	Stuttgart
Hendrik Jan Hoogeboom	Leiden
Paul Gastin	Paris
Wojtek Penczek	Warsaw
Antoine Petit	Orsay
Giovanni Pighizzini	Milano
Nicoletta Sabadini	Milano
Mike Stannett	Sheffield

Guests

Siegmar Gerber	Stuttgart/Leipzig
Anca Muscholl	Stuttgart
Klaus Reinhardt[1]	Stuttgart

[1] Not on the picture, since he took the photograph.

Synthesis of Nondeterministic Asynchronous Automata

Giovanni Pighizzini [†]
Dipartimento di Scienze dell'Informazione
Università degli Studi di Milano

Abstract

In this paper we give an algorithm for building a nondeterministic asynchronous automaton recognizing the trace language [Maz77] denoted by a given *recognizable expression*. Moreover, we show that the number of states of the automaton so obtained is polynomial in the length of the expression.

1 Introduction

Asynchronous Automata were introduced by W. Zielonka [Zie87] as algebraic abstactions of distributed systems. This kind of automata is very interesting since, as shown in [Zie87, CM88], they characterize the class of *recognizable trace languages* [BBMS81], that is the class of languages accepted by finite automata over free partially commutative monoids.

Another characterization of the class of recognizable trace languages were discovered by E. Ochmański [Och85], proving that the class of recognizable trace languages coincides with the minimal class containing finite trace languages and closed under union, product and a new operation called *concurrent iteration*.

In this paper we give an algorithm for building nondeterministic asynchronous automata accepting the union, the product and the concurrent

[†]Author's address: Dipartimento di Scienze dell'Informazione, Via Comelico 39, 20135 Milano, Italy – Email: pighizzi@imiucca.csi.unimi.it PP Supported in part by the ESPRIT Basic Research Action No. 3166: "Algebraic and Syntactic Methods in Computer Science (ASMICS)" and by MURST 40%.

iteration of trace languages accepted by given nondeterministic asynchronous automata. Our constructions are nontrivial generalizations of the corresponding constructions for finite automata (see for example [HopUll67]). The number of states of the resulting automaton is polynomial in the number of states of the given automata. The algorithm for building the asynchronous automaton accepting the concurrent iteration of the language accepted by a given asynchronous automaton is obtained using a new interesting result concerning the iteration of trace languages. More precisely, we prove that the iteration of a trace language T can be expressed as finite union of the iterations of suitable languages $T_1, \ldots, T_m$ such that in every language T_i, $1 \leq i \leq m$, all traces have *exactly the same set of letters*.

2 Traces and trace languages

In this section we briefly recall the basic notions concerning trace languages.

Definition 2.1 *A* concurrent alphabet *is a pair (A, θ), where A is a finite alphabet and $\theta \subseteq A \times A$ is a symmetric and irreflexive relation, called* independency relation. *If $(a, b) \in \theta$ then we will say that a and b are* independent *letters; otherwise a and b are* dependent. *$Cliques(A, \theta)$ denotes a (fixed) family of cliques covering the complement of θ.*

The free partially commutative monoid *(fpcm, for short) generated by (A, θ) is defined as the initial object in the category of monoids generated by A and satisfying, besides the usual monoid axioms, the set of "commutativity laws" $\{ab = ba \mid a \theta b\}$.*
A trace *is an element of $M(A, \theta)$. A* trace language *is a subset of $M(A, \theta)$.*

It is well–known that $M(A, \theta)$ is isomorphic to the quotient structure $A/\equiv_\theta$ where $\equiv_\theta$ is the least congruence over A^* extending the set of commutativity laws. Then, a trace is an equivalence class of words. The equivalence class containing the word $w \in A^*$ will be denoted as $[w]_\theta$ or by $[w]$ if θ is understood. A trace x is said to be a *prefix* (*suffix*) of a trace t if and only if there is a trace z such that $t = xz$ ($t = zx$, respectively).

The set of symbols of a trace $t \in M(A,\theta)$ will be denoted by $alph(t)$, i.e. $alph(\epsilon) = \emptyset$ and $alph(ta) = alph(t) \cup \{a\}$ for $t \in M(A,\theta)$, $a \in A$. Moreover, $alph(T) = \bigcup_{t\in T} alph(t)$ for every $T \subseteq M(A,\theta)$. A subset $\alpha \subseteq A$ is said to be *connected* if for $a, b \in \alpha$ we can find $a_0, a_1, \ldots, a_n \in \alpha$ such that $a_0 = a, a_n = b$ and $(a_{i-1}, a_i) \notin \theta$ for $i = 1, \ldots, n$. A trace t is *connected* if $alph(t)$ is connected. A language T is connected if and only if every trace in T is connected.

A trace u is a *component* of a trace t if u is connected and there exists a trace v such that $t = uv$ and $alph(u) \times alph(v) \subseteq \theta$. The set of all components of a trace t will be denoted as $c(t)$.

We are interested in the usual regular operations (union, product and iteration, denoted, as usual, with the symbols $\cdot$, $\cup$ and *, respectively), and in the operations on trace languages defined as follows:

Definition 2.2 *Let T be a trace language over the fpcm $M(A,\theta)$, and α a subset of A. The* decomposition *of T, denoted by $c(T)$ is the set of all components of traces in T, i.e. $c(T) = \bigcup_{t\in T} c(t)$. The* concurrent iteration (coiteration, *for short) of T is the language $T^{co-*} = c(T)^*$. The* α–restriction *of T is the set $T_\alpha = \{t \in T \mid alph(t) = \alpha\}$. Finally, the* restricted iteration *of T is the language $T^{\oplus}$ defined as follows:*

$$T^{\oplus} = \begin{cases} T^+ & \text{if } T = T_\alpha \text{ for some connected set } \alpha \subseteq A \\ \emptyset & \text{otherwise,} \end{cases}$$

where, as usual, T^+ denotes the language $T \cdot T^*$.

The class $\boldsymbol{Reg}(A,\theta)$ of *regular trace languages* over (A,θ) [Maz77] is defined as the smallest class containing the languages $\emptyset$, $\{[\epsilon]\}$, $\{[a]\}$, for all $a \in A$, and closed with respect to the regular operations. On the other hand, the class $\boldsymbol{Rec}(A,\theta)$ of *recognizable trace languages* over (A,θ) can be defined in a standard way using the notion of $M(A,\theta)$–automata [BBMS81]. It is known that if the independency relation θ is not empty then the class $\boldsymbol{Rec}(A,\theta)$ is properly included in the class $\boldsymbol{Reg}(A,\theta)$.

The following characterization of the class $\boldsymbol{Rec}(A,\theta)$ were obtained by Ochmański [Och85]:

Theorem 2.1 *For every concurrent alphabet (A, θ), the class $\mathbf{Rec}(A, \theta)$ of recognizable trace languages over (A, θ) is the smallest class of trace languages containing the languages $\emptyset$, $\{[\epsilon]\}$, $\{[a]\}$, for all $a \in A$, and closed under union, product and coiteration.*

As a consequence of the last theorem, every recognizable trace language can be denoted by a *recognizable expression*, i.e. an expression involving operators representing the operations of union, product and coiteration of trace languages.

Given a set S, the family of subsets of S will be denoted as $\mathcal{P}(S)$.

Lemma 2.1 *Let $P \subseteq \mathcal{P}(A)$ be a family of subsets of A and $T \subseteq M(A, \theta)$ a trace language. If T is recognizable then also the language $\bigcup_{\alpha \in P} T_\alpha$ is recognizable.*

As a consequence of this lemma, for every recognizable language T the following sets are recognizable: the α–restriction of T (T_α, for $\alpha \subseteq A$), the set of all connected traces in T, the set of all nonconnected traces in T, the set $\{t \in T \mid \gamma \subseteq alph(t) \subseteq \alpha\} = \bigcup_{\gamma \subseteq \beta \subseteq \alpha} T_\beta$, for $\gamma \subseteq \alpha \subseteq A$.

3 A result on the iteration of trace languages

In this section we state our main result concerning the iteration of trace languages.

Given a set $\alpha \subseteq A$, $\alpha \neq \emptyset$, a *composition* of α is an ordered partition of α, i.e. a sequence $p_1, \ldots, p_s$, $s \geq 1$, of nonempty subsets of α whose union is equal to α and such that $p_i \cap p_j = \emptyset$ for $i \neq j$. The set of all compositions of α will be denoted by $\Pi(\alpha)$.

Theorem 3.1 *Given a language $T \subseteq M(A, \theta)$ and a set $\alpha \subseteq A$, consider the language $Z \subseteq M(A, \theta)$ so defined:*

$$Z = \begin{cases} \bigcup_{(p_1, \ldots, p_s) \in \Pi(\alpha)} XY_{p_1} XY_{p_2} \ldots Y_{p_s} X, & \text{if } \alpha \neq \emptyset; \\ \{[\epsilon]\}, & \text{otherwise,} \end{cases}$$

where $X = \bigcup_{\beta \subset \alpha} (T^*)_\beta$, *and* $Y_{p_j} = \bigcup_{p_j \subseteq \beta \subseteq \alpha} T_\beta$, *for* $j = 1, \ldots, s$.
Then, $(T^*)_\alpha = Z^+$ *and* $alph(t) = \alpha$ *for every trace* $t \in Z$,*i.e.* $Z = Z_\alpha$.

Proof. Before of proving the Theorem we remark that in the definition of X the union operation is extended to every subset β *strictly* included in α.
For $\alpha = \emptyset$ the proof is trivial. Then, we suppose $\alpha \neq \emptyset$.

The inclusion $Z^+ \subseteq (T^*)_\alpha$ can be easily verified. The converse inclusion, is a consequence of the following lemma:

Lemma 3.1 *For every trace* $u \in (T^*)_\alpha$, *either* $u \in Z$ *or there are traces* u', u'' *such that* $u = u'u''$, $u' \in Z$, $|u''| < |u|$ *and* $u'' \in (T^*)_\alpha$.

Proof. Since $u \in (T^*)_\alpha$ and $\alpha \neq \emptyset$, we can find traces $u_1, \ldots, u_m \in T$, $m \geq 1$ such that $u = u_1 \ldots u_m$. Since $alph(u) = \alpha$, there is an index k, $1 \leq k \leq m$, such that $alph(u_1 \ldots u_k) = \alpha$ and $alph(u_1 \ldots u_{k-1}) \neq \alpha$. We consider the sequence $\alpha_1, \ldots, \alpha_k$ of subsets of α such that $\alpha_i = alph(u_i) \Leftrightarrow \bigcup_{j=1}^{i-1} \alpha_j$, for $j = 1, \ldots, k$.
Clearly, $\bigcup_{j=1}^{k} \alpha_j = \alpha$ and $\alpha_i \cap \alpha_j = \emptyset$, for $i \neq j$.
From the sequence $\alpha_1, \ldots, \alpha_k$, we extract the subsequence $\alpha_{j_1}, \ldots, \alpha_{j_s}$ by eliminating the empty subsets (observe that $j_s = k$). This sequence is a composition of α and $\alpha_{j_r} \subseteq alph(u_{j_r}) \subseteq \alpha$, for $r = 1, \ldots, s$. Then, $u_{j_r} \in Y_{\alpha_{j_r}}$.
By construction, the set $\beta = alph(u_{j_{r-1}+1} \ldots u_{j_r-1})$ is strictly included in α, for $r = 1, \ldots, s$, $j_0 = 0$. On the other hand, $u_{j_{r-1}+1} \ldots u_{j_r-1} \in T^*$. So:

$$u_{j_{r-1}+1} \ldots u_{j_r-1} \in (T^*)_\beta \subseteq \bigcup_{\beta \subset \alpha} (T^*)_\beta = X.$$

This permits us to conclude that $u_1 \ldots u_k \in XY_{\alpha_{j_1}} X \ldots XY_{\alpha_{j_s}}$ and, observing that $[\epsilon] \in X$, we have: $u_1 \ldots u_k \in XY_{\alpha_{j_1}} X \ldots XY_{\alpha_{j_s}} X \subseteq Z$. Now, we set $u' = u_1 \ldots u_k$ and $u'' = u_{k+1} \ldots u_m$. Observe that $|u'| > 1$ Clearly, $alph(u'') \subseteq \alpha$. If $alph(u'') = \alpha$ then $u'' \in (T^*)_\alpha$ and $|u''| < |u|$ and the statement is proved. On the other hand, if $alph(u'') \neq \alpha$ then $u'' \in (T^*)_{alph(u'')} \subseteq X$. Recalling that $u_1 \ldots u_k \in XY_{\alpha_{j_1}} X \ldots XY_{\alpha_{j_s}}$, it turns out that $u = u'u'' \in (XY_{\alpha_{j_1}} X \ldots XY_{\alpha_{j_s}})X \subseteq Z$. Then $u \in Z$. This concludes the proof of the lemma. ∎

Now, using the previous lemma, it is possible to show by induction on the length of traces $t \in M(A,\theta)$ that $t \in (T^*)_\alpha$ implies $t \in Z^+$. Then $(T^*)_\alpha \subseteq Z^+$. ∎

Corollary 3.1 *For every trace language $T \subseteq M(A,\theta)$ there are languages $T_1, \ldots, T_m$, $m \geq 0$, such that $T^* = \bigcup_{i=1}^{m} (T_i)^*$ and for every i $(i = 1, \ldots, m)$ there is a subset α of A for which $T_i = (T_i)_\alpha$.*

4 Asynchronous Automata

In this section we briefly recall the definition and some properties of Asynchronous Automata.

Definition 4.1 *A* nondeterministic asynchronous automaton (NAA) *with n processes, over the concurrent alphabet (A,θ), is a tuple $M = \big(P_1, \ldots, P_n, \{\delta_a\}_{a\in A}, I, F\big)$, where:*

- *for $i = 1, \ldots, n$, $P_i = (A_i, S_i)$ is the* ith process*, where S_i is its set of* local states *and A_i is its* local alphabet*, such that $\{A_1, \ldots, A_n\} = Cliques(A,\theta)$;*
- *let $Proc = \{1, \ldots, n\}$, and, for $a \in A$, let be $Dom(a) = \{i \in Proc \mid a \in A_i\}$ the set of (indices of) processes that execute a; then $\delta_a : \prod_{i\in Dom(a)} S_i \to \mathcal{P}(\prod_{i\in Dom(a)} S_i)$ is the* local transition function *associated to the letter a;*
- *let $S = \prod_{i\in Proc} S_i$ be the set of* global states*; then $I \subseteq S$ is the set of the* initial states *and $F \subseteq S$ is the set of* final states.

A global state will be denoted by a boldface letter, as $\boldsymbol{s}$, while its ith component as s_i; given a global state $\boldsymbol{s} \in S$ and a set of (indices of) processes $\alpha = \{i_1, \ldots, i_k\} \subseteq Proc$, we will denote by $\boldsymbol{s}_{|\alpha}$ the vector whose components correspond to local states in α, i.e. $\boldsymbol{s}_{|\alpha} = (s_{i_1}, \ldots, s_{i_k})$.

The *global transition function* $\Delta : S \times M(A,\theta) \to \mathcal{P}(S)$ of the automaton M is defined on the ground of local transition functions as follows. For every $\boldsymbol{s}, \boldsymbol{u} \in S$, $a \in A$, we have: $\boldsymbol{u} \in \Delta(\boldsymbol{s}, a)$ if and only

if $\boldsymbol{u}_{|Dom(a)} \in \delta_a(\boldsymbol{s}_{|Dom(a)})$ and $\boldsymbol{u}_{|\overline{Dom(a)}} = \boldsymbol{s}_{|\overline{Dom(a)}}$. The extension to traces can be obtained in a standard way.

The *language* accepted by the automaton M is defined as the set:

$$T(M) = \{t \in M(A,\theta) \mid \Delta(I,t) \cap F \neq \emptyset\}.$$

If $\#I = 1$ and $\#\left(\delta_a(s_{i_1},\ldots,s_{i_k})\right) \leq 1$ for every $(s_{i_1},\ldots,s_{i_k}) \in \prod_{i\in Dom(a)} S_i$, $a \in A$, the automaton M is said to be a *deterministic* asynchronous automaton (DAA).

It is immediate to verify that to every asynchronous automaton can be associated a finite state automaton recognizing the same trace language. Moreover, given a finite automaton over the the fpcm $M(A,\theta)$ it is possible to construct a DAA with the same concurrent alphabet, accepting the same language. This result, not at all obvious, were obtained by W. Zielonka.

Theorem 4.1 [Zie87] *The class of languages accepted by deterministic asynchronous automata over (A,θ) coincides with the class $\boldsymbol{Rec}(A,\theta)$ of trace languages recognized by $M(A,\theta)$–automata.*

Another proof of Theorem 4.1 can be found in [CM88].

From this point on, we consider a fixed concurrent alphabet (A,θ) with a fixed clique cover $Cliques(A,\theta) = \{A_1,\ldots,A_n\}$, of the dependency relation. For all notations concerning NAA the reader is referred to this section. When we deal with different NAA M, M' and M'', we always suppose that $A_i = A'_i = A''_i$ and $S'_i \cap S''_i = \emptyset$ for $i \in Proc$.

4.1 Asynchronous automata with ϵ–moves

Here, we consider an extension of NAA useful in our algorithm. In the model we will consider every process can change its internal state, independently from the states of other processes and from the next input symbol, using an internal transition called ϵ–move. The ϵ–moves are represented extending every local alphabet with a new symbol. More

precisely, we consider the concurrent alphabet $(\widehat{A}, \widehat{\theta})$ such that $\widehat{A} = A \cup \{e_1, \ldots, e_n\}$, where $\{e_1, \ldots, e_n\} \cap A = \emptyset$, and

$$\widehat{\theta} = \theta \cup \{(e_i, e_j) \mid i \neq j\} \cup \{(a, e_i), (e_i, a) \mid a \in A \Leftrightarrow A_i\}.$$

Then $Cliques(\widehat{A}, \widehat{\theta}) = \{\widehat{A}_1, \ldots, \widehat{A}_n\}$, where $\widehat{A}_i = A_i \cup \{e_i\}$, $i = 1, \ldots, n$.

To every trace $\hat{t} \in M(\widehat{A}, \widehat{\theta})$ we associate the trace $\psi(\hat{t}) \in M(A, \theta)$ obtained erasing from $\hat{t}$ all letters not belonging to A. Given a language $\widehat{T} \subseteq M(\widehat{A}, \widehat{\theta})$ we denote by $\psi(\widehat{T})$ the set $\bigcup_{\hat{t} \in \widehat{T}} \psi(\hat{t})$.

The following theorem states an important relationship between NAA over $M(\widehat{A}, \widehat{\theta})$ and NAA over $M(A, \theta)$.

Theorem 4.2 *Let $\widehat{T} \subseteq M(\widehat{A}, \widehat{\theta})$ be the language accepted by a NAA $\widehat{M}$. Then, there exists a NAA over $M(A, \theta)$ accepting the language $\psi(\widehat{T})$.*

Proof (outline). The automaton M is defined as follows: $S_i = \widehat{S}_i$, $i = 1, \ldots, n$;
$I = \{\boldsymbol{s} \in S \mid \exists w \in M(\widehat{A}, \widehat{\theta})$ s.t. $alph(w) \subseteq \{e_1, \ldots, e_n\}$ and $\boldsymbol{s} \in \widehat{\Delta}(\widehat{I}, w)\}$;
$F = \widehat{F}$;
for $a \in A$ with $Dom(a) = \{i_1, \ldots, i_k\}$, $s \in \prod_{i \in Dom(a)} S_i$:

$$\delta'_a(s) = \{u \in \prod_{i \in Dom(a)} S_i \mid \exists \boldsymbol{s}', \boldsymbol{u}' \in \widehat{S} \text{ s.t. } \boldsymbol{s}'_{|Dom(a)} = s,\ \boldsymbol{u}'_{|Dom(a)} = u,$$
$$\exists w \in M(\widehat{A}, \widehat{\theta}) \text{ s.t. } alph(w) \subseteq \{e_{i_1}, \ldots, e_{i_k}\}, \text{ and } \boldsymbol{u}' \in \widehat{\Delta}(\boldsymbol{s}', aw)\}.$$

It is possible to prove that for $t \in M(A, \theta)$ it holds: $\Delta(I, t) = \bigcup_{\hat{t} \in \psi^{-1}(t)} \widehat{\Delta}(I, \hat{t})$. ∎

5 Synthesis of Asynchronous Automata

In this section we give an algorithm for building nondeterministic asynchronous automata from recognizable expressions.

First, we observe that asynchronous automata accepting the languages $\emptyset$, $\{[\epsilon]\}$ and $\{[a]\}, a \in A$, can be trivially constructed.

5.1 Union

The construction of a NAA M accepting the union of the languages accepted by two given NAA M' and M'' is trivial. Then, we give it without any comment.

We define M as follows: $S_i = S'_i \cup S''_i$, $i = 1, \ldots, n$; $I = I' \cup I''$; $F = F' \cup F''$; for $a \in A$ and $s \in \prod_{i \in Dom(a)} S_i$:

$$\delta_a(s) = \begin{cases} \delta'_a(s) & \text{if } s \in \prod_{i \in Dom(a)} S'_i \\ \delta''_a(s) & \text{if } s \in \prod_{i \in Dom(a)} S''_i \\ \emptyset & \text{otherwise.} \end{cases}$$

It is easy to verify that for every global state $\boldsymbol{s} \in I$ and for every trace $t \in M(A, \theta)$ $\Delta(\boldsymbol{s}, t)$ is equal to $\Delta'(\boldsymbol{s}, t)$ $(\Delta''(\boldsymbol{s}, t))$ if $\boldsymbol{s} \in I'$ $(\boldsymbol{s} \in I''$, respectively). So, M can simulate the computations of M' and M'' and it recognizes the language $T(M') \cup T(M'')$.

5.2 Product

Given two NAA M' and M'', for every $s \in S'$, let M'_s be the NAA $M'_s = (P'_1, \ldots, P'_n, \{\delta'_a\}_{a \in A}, I', \{s\})$ and for every $s \in S''$ let ${}_sM''$ be the NAA ${}_sM'' = (P''_1, \ldots, P''_n, \{\delta''_a\}_{a \in A}, \{s\}, F'')$. Then $T(M') \cdot T(M'') = \cup_{(s', s'') \in F' \times I''} T(M'_{s'} \cdot T({}_{s''}M''))$. This observation permit us of *considering in this section only automata M' and M'' such that $\#F' = 1$ and $\#I'' = 1$.* We will denote by $\boldsymbol{F}' = (F'_1, \ldots, F'_n)$ the only final state of M' and by $\boldsymbol{I}'' = (I''_1, \ldots, I''_n)$ the only initial state of M''.

We define the following automaton M with ϵ–moves.

- $S_i = S'_i \cup S''_i$, $i = 1, \ldots, n$;
- $I = I'$;
- $F = F''$;
- for $a \in A$ with $Dom(a) = \{i_1, \ldots, i_k\}$, $s_{i_j} \in S_{i_j}$, $j = 1, \ldots, k$:

$$\delta_a(s_{i_1}, \ldots, s_{i_k}) = \begin{cases} \delta'_a(s_{i_1}, \ldots, s_{i_k}) & \text{if } s_{i_j} \in S'_{i_j}, 1 \leq j \leq k, \\ \delta''_a(s_{i_1}, \ldots, s_{i_k}) & \text{if } s_{i_j} \in S''_{i_j}, 1 \leq j \leq k, \\ \emptyset & \text{otherwise;} \end{cases}$$

- for $i = 1, \ldots, n$, $s_i \in S_i$:

$$\delta_{e_i}(s_i) = \begin{cases} \{I_i''\} & \text{if } s_i = F_i', \\ \emptyset & \text{otherwise.} \end{cases}$$

Now, we prove that $T(M) = T(M')\{[e_1 \ldots e_n]\}T(M'')$.
First, we get the following immediate remarks.

Remarks

(a) For every trace $t \in M(A, \theta)$ (i.e., T does not contain any symbol corresponding to an ϵ–move) and for every $\boldsymbol{s} \in S'$: $\Delta(\boldsymbol{s}, t) = \Delta'(\boldsymbol{s}, t)$;

(b) for every trace $t \in M(A, \theta)$ and for every $\boldsymbol{s} \in S''$: $\Delta(\boldsymbol{s}, t) = \Delta''(\boldsymbol{s}, t)$;

(c) $\Delta(\boldsymbol{F}', [e_1 \ldots e_n]) = \boldsymbol{I}''$;

(d) for every $t \in T(M)$ every process P_i on input t executes exactly one ϵ-move, i.e. $|t|_{e_i} = 1$.

Now, we can prove the following theorem.

Theorem 5.1 $T(M) = T(M')\{e_1 \ldots e_n\}T(M'')$

Proof. The inclusion of $T(M')\{[e_1 \ldots e_n]\}T(M'')$ in $T(M)$ is an immediate consequence of previous remarks. In order to prove the converse inclusion, we consider a trace $t \in T(M)$. From Remark (d), we can find traces $u_0, \ldots, u_n \in M(A, \theta)$ and a permutation $(i_1, \ldots, i_n)$ of $(1, \ldots, n)$ such that $t = u_0 e_{i_1} u_1 e_{i_2} \ldots e_{i_n} u_n$.
Let be $1 \leq k \leq n$ and $a \in alph(u_k)$. Then $u_k = vaw$ for suitable traces $v, w \in M(A, \theta)$ and $t = u_0 \ldots e_{i_k} vawe_{i_{k+1}} \ldots u_n$. Since M accepts t, there are global states $\boldsymbol{s}', \boldsymbol{s}'' \in S$ such that:

$\boldsymbol{s}' \in \Delta(I', u_0 \ldots e_{i_k} v)$,

$\boldsymbol{s}'' \in \Delta(\boldsymbol{s}', a)$ and

$\Delta(\boldsymbol{s}'', we_{i_{k+1}} \ldots u_n) \cap F \neq \emptyset$.

More precisely, we can observe that both local states s'_{i_j}, s''_{i_j} belong to S'_{i_j}, for $j > k$, while both s'_{i_j}, s''_{i_j} belong to S''_{i_j} for $j \leq k$. Then, from the definition of δ_a and from $\Delta(\boldsymbol{s}', a) \neq \emptyset$ it follows that either $Dom(a) \subseteq \{i_{k+1}, \ldots, i_n\}$ or $Dom(a) \subseteq \{i_1, \ldots, i_k\}$. So, we can decompose u_k as product of two independent traces u'_k and u''_k, s.t. $Dom(u'_k) \subseteq$

$\{i_{k+1}, \ldots, i_n\}$ and $Dom(u''_k) \subseteq \{i_1, \ldots, i_k\}$. Moreover $Dom(u''_k) \cap Dom(u'_j) = \emptyset$ for $j > k$. Then, we can rewrite t as $u'_0 \ldots u'_n[e_1 \ldots e_n]u''_0 \ldots u''_n$. Finally, let $\boldsymbol{s} \in I'$, $\boldsymbol{q} \in F''$ be the states corresponding to the start and to the end of such an accepting computation. More precisely:

$\boldsymbol{s'} \in \Delta(\boldsymbol{s}, u_0 \ldots e_{i_k} v)$ and $\boldsymbol{q} \in \Delta(\boldsymbol{s''}, we_{i_{k+1}} \ldots u_n)$.

It is easy to observe that:

$\boldsymbol{F'} \in \Delta'(\boldsymbol{s}, u'_0 \ldots u'_n)$ and $\boldsymbol{q} \in \Delta''(\boldsymbol{I''}, u''_0 \ldots u''_n)$.

Then $u'_0 \ldots u'_n \in T(M')$ and $u''_0 \ldots u''_n \in T(M'')$. ∎

Finally, applying the construction given in the proof of Theorem 4.2, we can obtain a NAA accepting the language $T(M')T(M'')$.

5.3 Concurrent iteration

In this section, we will show how it is possible to construct a NAA recognizing the coiteration of the language recognized by a given NAA. To achieve this goal, we will prove that the coiteration T^{co-*} of a trace language T can be expressed by means of the operations of union, product, decomposition, α–restriction and restricted iteration, applied starting from T, and we will give algorithms for building asynchronous automata accepting the decomposition, the α–restriction and the restricted iteration of the language accepted by a given NAA.

First, we recall that $T^{co-*} = (c(T))^* = \bigcup_{\alpha \subseteq A}((c(T))^*)_\alpha$ and we prove that every language $(T^{co-*})_\alpha = ((c(T))^*)_\alpha$ can be expressed inductively from T and $(T^{co-*})_\beta$, where β is strictly included in α, using the above mentioned operations.

For $\alpha = \emptyset$, we have $(T^{co-*})_\emptyset = \{[\epsilon]\}$.

Suppose now that $\alpha \neq \emptyset$. If α is nonconnected, we have $(T^{co-*})_\alpha = ((c(T))^+)_{\alpha_1} \cdot \ldots \cdot ((c(T))^+)_{\alpha_k}$, where the sets $\alpha_1, \ldots, \alpha_k$ are the connected components of α; then $((c(T))^+)_{\alpha_j} = (T^{co-*})_{\alpha_j}$, with $\alpha_j \subset \alpha$, $j = 1, \ldots, k$.

On the other hand, if α is connected, then by Theorem 3.1 we have: $(T^{co-*})_\alpha = Z^+$, where $Z = \bigcup_{(p_1,\ldots,p_s) \in \Pi(\alpha)} XY_{p_1}XY_{p_2} \ldots Y_{p_s}X$, $X = \bigcup_{\beta \subset \alpha}((c(T))^*)_\beta = \bigcup_{\beta \subset \alpha}(T^{co-*})_\beta$, and, for $j = 1, \ldots, s$, $Y_{p_j} =$

$\bigcup_{p_j \subseteq \beta \subseteq \alpha} c(T)_\beta$. Moreover, $alph(t) = \alpha$, for every trace $t \in Z$. Then $Z^+ = Z^\oplus$. This implies that $(T^{co-*})_\alpha = Z^\oplus$.

As a consequence of this discussion, it is easy to obtain the following new characterization of the class of recognizable trace languages:

Theorem 5.2 *The class* $\boldsymbol{Rec}(A, \theta)$ *of recognizable trace languages over* (A, θ) *is the smallest class containing the languages* $\emptyset$, $\{[\epsilon]\}$, *and* $\{[a]\}$ *(for* $a \in A$*) and closed with respect to the operations of union, product, decomposition, restricted iteration and* α*–restriction.*

Now, we have to give the constructions of NAA accepting the α–restriction, the decomposition and the restricted iteration of the trace language accepted by a given NAA.

5.3.1 Restriction

Here, we briefly describe how it is possible to build a NAA accepting the restriction of the language accepted by a given NAA. The idea is essentialy that of keeping in every local state the alphabet of the trace executed so far. More precisely, we consider the set of local states $S'_i = S_i \times \mathcal{P}(A_i)$, $i \in Proc$, the set of initial global states $I' = \{((s_1, \emptyset), \ldots, (s_n, \emptyset)) \mid \boldsymbol{s} \in I\}$, and the local transition function δ_a, $a \in A$, such that:

$$\begin{aligned}\delta'_a((s_{i_1}, \alpha_{i_1}), \ldots, (s_{i_k}, \alpha_{i_k})) = \\ \{((u_{i_1}, \alpha_{i_1} \cup \{a\}), \ldots, (u_{i_k}, \alpha_{i_k} \cup \{a\})) \mid \\ (u_{i_1}, \ldots, u_{i_k}) \in \delta_a(s_{i_1}, \ldots, s_{i_k})\},\end{aligned}$$

for $s_{i_j} \in S_{i_j}$, $\alpha_{i_j} \subseteq A_{i_j}$, $j = 1, \ldots, k$, where $\{i_1, \ldots, i_k\} = Dom(a)$.

It is not difficult to verify that for every trace $t \in M(A, \theta)$, it holds:

$$\Delta'(I', t) = \{((s_1, alph(t) \cap A_1), \ldots, (s_n, alph(t) \cap A_n)) \mid \boldsymbol{s} \in \Delta(I, t)\}.$$

We now define two NAAs $M_k = (P'_1, \ldots, P'_n, \{\delta'_a\}, I', F_k)$, $k = 1, 2$.

M_1 is obtained choosing as set of final states the set:

$$F_1 = \{((s_1, \alpha_1), \ldots, (s_n, \alpha_n)) | (s_1, \ldots, s_n) \in F\}.$$

Of course, the language accepted by M_1 coincides with the language accepted by M, i.e., $T(M_1) = T(M)$. Fixed a set $\alpha \subseteq A$, in order to recognize the α–restriction of $T(M)$ we have to accept all traces of $T(M)$ whose alphabet is exactly α. This goal can be achieved setting

$$F_2 = \{((s_1, \alpha_1), \ldots, (s_n, \alpha_n)) | (s_1, \ldots, s_n) \in F \text{ and } \bigcup_{i=1}^{n} \alpha_i = \alpha\}.$$

It is clear that $T(M_2) = (T(M))_\alpha$.

5.3.2 Decomposition

The technique above explained for the construction of a NAA accepting the restriction of the language accepted by a given NAA, can be extended in order to obtain a NAA for the decomposition.

Given a NAA M, we define the sets of local states $S_i' = S_i \times \mathcal{P}(A_i) \times S_i$, $i = 1, \ldots, n$, the set of initial global states $I'' = \{((r_1, \emptyset, r_1), \ldots, (r_n, \emptyset, r_n)) \mid \boldsymbol{r} \in I\}$, and the local transition function, for $a \in A$, with $Dom(a) = \{i_1, \ldots, i_k\}$, $s_{i_j}, r_{i_j} \in S_{i_j}$, $\alpha_{i_j} \subseteq A_{i_j}$, $j = 1, \ldots, k$:

$$\begin{aligned}\delta_a''((s_{i_1}, \alpha_{i_1}, r_{i_1}), \ldots, (s_{i_k}, \alpha_{i_k}, r_{i_k})) = \\ \{((u_{i_1}, \alpha_1 \cup \{a\}, r_{i_1}), \ldots, (u_{i_k}, \alpha_k \cup \{a\}, r_{i_k})) \mid \\ (u_{i_1}, \ldots, u_{i_k}) \in \delta_a(s_{i_1}, \ldots, s_{i_k})\}.\end{aligned}$$

Then, for every $t \in M(A, \theta)$ and for every $\boldsymbol{r} \in S$:

$$\begin{aligned}&\Delta''(((r_1, \emptyset, r_1), \ldots, (r_n, \emptyset, r_n)), t) = \\ &\quad = \{((s_1, alph(t) \cap A_1, r_1), \ldots, (s_n, alph(t) \cap A_n, r_n)) \mid \boldsymbol{s} \in \Delta(\boldsymbol{r}, t)\}.\end{aligned}$$

Let now t be a trace in the language $T(M) = T(M_1)$ and t' a connected component of t, i.e., $t = t't''$, $alph(t')$ is connected and $alph(t') \times alph(t'') \subseteq \theta$, for some $t'' \in M(A, \theta)$. Then $t' \in c(T(M))$. Since t is accepted by M, we can find global states $\boldsymbol{u} \in F$, $\boldsymbol{r} \in I$ such that $\boldsymbol{u} \in \Delta(\boldsymbol{r}, t)$. So:

$$\begin{aligned}((u_1, alph(t) \cap A_1, r_1), \ldots, (u_n, alph(t) \cap A_n, r_n)) \in \\ \Delta''(((r_1, \emptyset, r_1), \ldots, (r_n, \emptyset, r_n)), t).\end{aligned}$$

Let be , $= Dom(t')$ and, for $i = 1, \ldots, n$:

$$(s_i, \alpha_i, r_i) = \begin{cases} (u_i, alph(t) \cap A_i, r_i) & \text{if } i \in , , \\ (r_i, \emptyset, r_i) & \text{otherwise.} \end{cases}$$

It is possible to verify that:

$$((s_1, \alpha_1, r_1), \ldots, (s_n, \alpha_n, r_n)) \in \Delta'(((r_1, \emptyset, r_1), \ldots, (r_n, \emptyset, r_n)), t'),$$

the set $\bigcup_{i=1}^{n} \alpha_i = alph(t')$ is connected and $\alpha_i \times (alph(t) \cap A_j) \subseteq \theta$, for $i \in , , j \in Proc \Leftrightarrow , .$ Moreover, it turns out that $alph(t''') = alph(t')$, $t'''t'' \in T(M)$ and $t''' \in c(T(M))$, for every trace t''' such that $((s_1, \alpha_1, r_1), \ldots, (s_n, \alpha_n, r_n)) \in \Delta''(((r_1, \emptyset, r_1), \ldots, (r_n, \emptyset, r_n)), t''')$.

From this discussion, we can conclude that the NAA $M'' = (P_1'', \ldots, P_n'', \{\delta_a''\}, I'', F'')$, where F'' is the set:

$$\begin{aligned} F'' = \{ & ((s_1, \alpha_1, r_1), \ldots, (s_n, \alpha_n, r_n)) \mid \\ & \boldsymbol{r} \in I, \exists\ \boldsymbol{u} \in F,\ t \in M(A, \theta) \text{ s.t. } \boldsymbol{u} \in \Delta(\boldsymbol{r}, t), \\ & \exists , \subseteq Proc \text{ s.t. } \forall i \in ,\ \ s_i = u_i, \alpha_i = alph(t) \cap A_i, \\ & \forall i \notin ,\ \ s_i = r_i, \alpha_i = \emptyset, \\ & \text{the set } \bigcup_{i=1}^{n} \alpha_i \text{ is connected and} \\ & \alpha_i \times (alph(t) \cap A_j) \subseteq \theta, \text{for } i \in , , \ j \in Proc \Leftrightarrow , \}, \end{aligned}$$

recognizes the decomposition of $T(M)$.

Remark
The condition $\exists t \in M(A, \theta)$ s.t. $\boldsymbol{u} \in \Delta(\boldsymbol{r}, t)$ is useful to assure that *only* connected components of traces of $T(M)$ are accepted. In fact, it can happen that there is none trace t such that $\boldsymbol{u} \in \Delta(\boldsymbol{r}, t)$, i.e. $\boldsymbol{u}$ is not reachable from $\boldsymbol{r}$, but $((s_1, \alpha_1, r_1), \ldots, (s_n, \alpha_n, r_n)) \in \Delta'(((r_1, \emptyset, r_1), \ldots, (r_n, \emptyset, r_n)), t')$ for some $t' \in M(A, \theta) \Leftrightarrow c(T(M))$, and the global state $((s_1, \alpha_1, r_1), \ldots, (s_n, \alpha_n, r_n))$ satisfies the other requirements of the definition of F''.

5.3.3 Restricted iteration

First, we observe that given a NAA M accepting a trace language $T \subseteq M(A,\theta)$ we can decide if $T = T_\alpha$ for some connected set $\alpha \subseteq A$, by constructing a finite state automaton M accepting T and verifying if M is equivalent to some automaton M_α accepting the language T_α for a connected $\alpha \subseteq A$.

Since the asynchronous automaton recognizing the empty language can be trivially constructed, from this point on we will consider the problem of building an asynchronous automaton accepting $T^{\oplus} = T^{+}$, when $T = T_\alpha$ for some connected alphabet $\alpha \subseteq A$.

Let M be a NAA accepting a trace language T, such that $T = T_\alpha$, for some connected set $\alpha \subseteq A$. There is none loss of generality assuming that the following conditions hold:

1. $\alpha = A$;
2. there exists a map $\phi : \bigcup_{i=1}^{n} S_i \to \mathcal{P}(A)$ such that if $\Delta(\boldsymbol{I}, t) = \boldsymbol{s}$ then $\phi(s_i) = alph(t) \cap A_i$, for $i = 1, \ldots, n$, $t \in M(A,\theta)$ (this means that the automaton M records the letters of traces. Observe that if s_i is a compontent of some final state $\boldsymbol{p}$, then $\phi(s_i) = A_i$).

Suppose in fact that $\alpha \subset A$. From the asynchronous automaton M it is possible to obtain an automaton $\widetilde{M}$ by removing all the processes P_i such that $A_i \cap \alpha = \emptyset$ and all transitions involving letters not belonging to α. Applying the construction presented below, we will obtain an automaton $\widetilde{M}'$ over the concurrent alphabet $(\alpha, \theta \cap (\alpha \times \alpha))$. This automaton can be easily extended to the concurrent alphabet (A,θ), by introducing dummy processes and null transition functions.
We observe that for the automaton M_1 defined in Section 5.3.1, the second condition holds.

For sake of simplicity, first we suppose that $\#I = \#F = 1$. Then, we will denote by $\boldsymbol{I} = (I_1, \ldots, I_n)$ and by $\boldsymbol{F} = (F_1, \ldots, F_n)$ the only initial and the only final state of M, respectively.

In the classical construction of the finite automaton N' recognizing the iteration of the language recognized by a given finite automaton N

(see e.g. [HopUll67]), an empty transition from the final state to the initial state is added to N. In our case, we want to construct an automaton M' recognizing all traces of the form $t_1[e_1 \dots e_n]t_2[e_1 \dots e_n] \dots [e_1 \dots e_n]t_m$ such that $m > 0$ and $t_i \in T(M)$, $i = 1, \dots, m$. However, as shown in the following example, the immediate generalization to asynchronous automata of the strategy used for finite automata does not work.

Example
Let (A, θ) be the concurrent alphabet with:

$$A = \{a, b, c, d\},\ \theta = \{(a, c), (c, a), (a, d), (d, a), (b, d), (d, b)\},$$

and $Cliques(A, \theta) = \{A_1, A_2, A_3\}$, where $A_1 = \{a, b\}$, $A_2 = \{b, c\}$ and $A_3 = \{c, d\}$. Let M be the asynchronous automaton represented in Fig. 1. It is not difficult to see that M accepts the set $T(M) = \{[acbd]\}$.

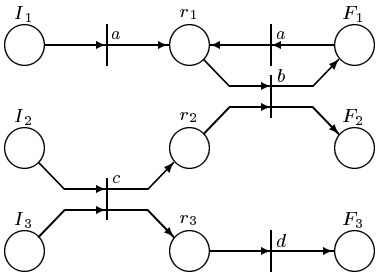


Figure 1: automaton M

Moreover, M satisfies the hypothesis stated above (that is $\alpha = A$ and there exists the map ϕ). Now, if we add, for every $i \in Proc$, a transition from the component F_i of the final state to the component I_i of the initial state, the resulting automaton can recognize a set that *properly* includes $(T(M))^+$. The automaton obtained applying this procedure in our specific case, is represented in Figure 2. Such an automaton accepts, for example, the trace $[acbade_2e_3cbd]$. Observe that $Dom(b) = \{1, 2\}$,

but before the last occurrence of b, the process P_2 executed one ϵ–move, while none ϵ–move was execute by P_1.

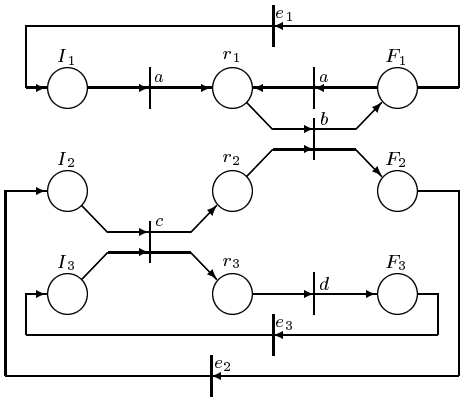


Figure 2: automaton constructed introducing an ϵ–move from every component of the final state to the corresponding component of the initial state

We will show that, in order to avoid the problems arising in previous example, we have to circularly rearrange two isomorphic copies of the given NAA M.

Formally, we define M' in the following way:

- $S'_i = S_i \times \{0,1\}$, $i = 1, \ldots, n$;
- $I' = \{((I_1, 0), \ldots, (I_n, 0))\}$;
- for $a \in A$ with $Dom(a) = \{i_1, \ldots, i_k\}$, $s_{i_j} \in S_{i_j}$, $j = 1, \ldots, k$, $b_{i_1}, \ldots, b_{i_k} \in \{0,1\}$:

$$\delta'_a((s_{i_1}, b_{i_1}), \ldots, (s_{i_k}, b_{i_k})) = \\ = \{((u_{i_1}, b), \ldots, (u_{i_k}, b)) \mid (u_{i_1}, \ldots, u_{i_k}) \in \delta_a(s_{i_1}, \ldots, s_{i_k})\},$$

if there exists $b \in \{0,1\}$ s.t. $b_{i_1} = \ldots = b_{i_k} = b$;

$$\delta'_a((s_{i_1}, b_{i_1}), \ldots, (s_{i_k}, b_{i_k})) = \emptyset$$

otherwise.

- for $i = 1, \ldots, n$, $s_i \in S_i$, $b \in \{0,1\}$:

$$\delta'_{e_i}(s_i, b) = \begin{cases} \{(I_i, 1 \Leftrightarrow b)\} & \text{if } s_i = F_i \\ \emptyset & \text{otherwise;} \end{cases}$$

- $F' = \{((F_1, b), \ldots, (F_n, b)) \mid b \in \{0,1\}\}$.

In Figure 3, the automaton M' obtained applying this costruction to the automaton M of Figure 1, is represented.

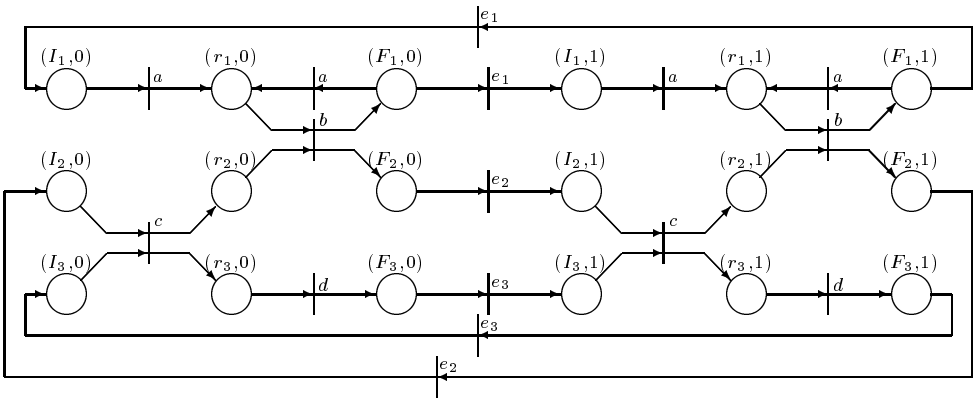


Figure 3: automaton M'

To prove the correctness of the construction, we will show that the automaton M' so defined recognizes the set of the traces of the form $t_1[e_1 \ldots e_n]t_2[e_1 \ldots e_n] \ldots [e_1 \ldots e_n]t_m$, where $m \geq 1$ and $t_i \in T$, $i = 1, \ldots, m$.

It is not difficult to see that every trace of such a form is accepted by M'. Then, we prove the converse result.

First, we will prove the following remarks:

Remarks
(a) Between two ϵ–moves of a process P'_i in an accepting computation, there is at least one occurence of every letter of A_i.

(b) In every accepting computation of M', before of the execution of a symbol a there is either at least one ϵ–move of every process in $Dom(a)$ or none ϵ–move of processes in $Dom(a)$.

(c) If some process of M' performs an ϵ–move on an accepted trace then *every* process performs an ϵ–move.

Remarks (a),(b) and (c) are consequences of the following lemmas, respectively.

Lemma 5.1 *If $t \in (\widehat{A}, \widehat{\theta})$ is accepted by M' and $t = t_0 e_i t_1 e_i t_2$, for suitable traces $t_0, t_1, t_2 \in (\widehat{A}, \widehat{\theta})$, then $A_i \subseteq alph(t_0)$, $A_i \subseteq alph(t_1)$, $A_i \subseteq alph(t_2)$.*

Proof. We prove that $A_i \subseteq alph(t_1)$. The other proofs are very similar. First, we suppose that $e_i \notin alph(t_1)$. Since $t \in T(M')$, we can find suitable global states of M' such that:
$((s_1, b_1), \ldots, (s_n, b_n)) \in \Delta'(I', t_0 e_i)$,
$((u_1, b'_1), \ldots, (u_n, b'_n)) \in \Delta'(((s_1, b_1), \ldots, (s_n, b_n)), t_1)$ and
$\Delta'(((u_1, b'_1), \ldots, (u_n, b'_n)), e_i t_2) \cap F' \neq \emptyset$.
Since $e_i \notin alph(t_1)$, we have $b_i = b'_i$. Moreover (s_i, b_i) is reached after one ϵ–move of the process P_i. Then $s_i = I_i$ and $\phi(s_i) = \emptyset$.
Since $\delta_{e_i}(u_i, b'_i) \neq \emptyset$, we have $u_i = F_i$. Then $\phi(u_i) = A_i$. This implies that $A_i \subseteq alph(t_1)$.

If $e_i \in alph(t_1)$, then there are traces $v_1, \ldots, v_m$ such that $t_1 = v_1 e_i v_2 e_i \ldots e_i v_m$ and $e_i \notin alph(v_j)$, $j = 1, \ldots, m$. Using the previous argument it can be shown that $A_i \subseteq alph(v_j)$. ∎

Lemma 5.2 *Let $t \in M(\widehat{A}, \widehat{\theta})$ be a trace accepted by the automaton M', with $t = u e_i v a w$, for some $u, v, w \in M(\widehat{A}, \widehat{\theta})$, $a \in A$, $i \in Proc$, such that $e_i \notin alph(u)$, and let $j \in Proc$ be an index such that $e_j \notin alph(u e_i v)$. Then $\{i, j\}$ cannot be a subset of $Dom(a)$.*

Proof. Since $t \in T(M')$, there are $\boldsymbol{s}, \boldsymbol{s}', \boldsymbol{s}'' \in S$, $b_1, \ldots, b_n, b'_1, \ldots, b'_n$, $b''_1, \ldots, b''_n \in \{0,1\}$ such that:
$((s_1, b_1), \ldots, (s_n, b_n)) \in \Delta'(((I_1, 0), \ldots, (I_n, 0)), ue_i)$,
$((s'_1, b'_1), \ldots, (s'_n, b'_n)) \in \Delta'(((s_1, b_1), \ldots, (s_n, b_n)), v)$,
$((s''_1, b''_1), \ldots, (s''_n, b''_n)) \in \Delta'(((s'_1, b'_1), \ldots, (s'_n, b'_n)), a)$,
$\Delta'(((s''_1, b''_1), \ldots, (s''_n, b''_n)), w) \cap F' \neq \emptyset$.
If both i, j belong to $Dom(a)$ then from $\Delta(((s'_1, b'_1), \ldots, (s'_n, b'_n)), a) \neq \emptyset$ and from $e_j \notin alph(uv)$, it turns out that $b'_i = b'_j = 0$. On the other hand, observing that after the ϵ–move of the process P_i it holds $b_i = 1$ and $s_i = I_i$, we can conclude that $e_i \in alph(v)$, i.e. $v = z' e_i z''$ for suitable traces z' and z''. From Lemma 5.1 we have $a \in alph(z')$ and $|v| \neq 0$. Then $t = ue_i v_1 a w_1$, for suitable traces v_1, w_1, with $|v_1| < |v|$. We can iterate the proof for this decomposition. So, we will find a countable sequence of traces $v_1, v_2, \ldots$ such that for every k $|v_k| < |v_{k-1}|$ and $|v_k| \neq 0$. This is contradictory. Then $Dom(a)$ cannot contain both indices i and j. ∎

Lemma 5.3 *Let $t \in M(\widehat{A}, \widehat{\theta})$ be a trace accepted by the automaton M'. If there is an index $i \in Proc$ such that $e_i \in alph(t)$, then $e_j \in alph(t)$, for every $j \in Proc$.*

Proof. First, we prove the result for $A_i \cap A_j \neq \emptyset$. Since t is accepted, there exists a global state $((p_1, b), \ldots, (p_n, b))$ belonging to $\Delta'(I', t) \cap F'$. Suppose that $e_j \notin alph(t)$. This means that the component P_j does not perform ϵ–moves and then $b = 0$. On the other hand $e_i \in alph(t)$. Then, this ϵ–move of the component P_i changes the last component of the local state of P_i from 0 to 1. So, at least another ϵ–move must be performed by P_i in order to have $b = 0$. Then, we can decompose t as $t = t_0 e_i t_1 e_i t_2$. As a consequence of Lemma 5.1, for every a letter $a \in A_i \cap A_j$, we have $a \in alph(t_1)$. From Lemma 5.2, it turns out that $\{i, j\}$ cannot be a subset of $Dom(a)$. This is a contradiction. Then we conclude that $e_i \in alph(t)$.

To prove the lemma in the general case, we recall that the alphabet A is connected. Then, we can find a sequence $P_{i_0}, \ldots, P_{i_m}$ of processes such that $i_0 = i$, $i_m = j$ and $A_{i_{k-1}} \cap A_{i_k} \neq \emptyset$, $k = 1, \ldots, m$. The

proof of the lemma follows applying the previous argument to every pair (i_{k-1}, i_k). ∎

Now, we can prove the correctness of the construction stated here.

Theorem 5.3 *Let M' the NAA above defined. Then:*

$$T(M') = \{t_1[e_1 \dots e_n]t_2 \dots t_{m-1}[e_1 \dots e_n]t_m \mid m \geq 1 \quad \textit{and } t_1, \dots, t_m \in T(M)\}.$$

Proof. The difficult point is to prove that every trace accepted by the automaton M' has the form $t_1[e_1 \dots e_n]t_2 \dots t_{m-1}[e_1 \dots e_n]t_m$, with $m \geq 1$, $t_i \in T$, $i = 1, \dots, m$. First, we observe that every trace t accepted by M' without ϵ-moves, i.e. $alph(t) \subseteq A$, is accepted also by the automaton M. Then, $t \in T(M)^+$. Suppose now that t is accepted using at least one ϵ-move. From Remark (c) it turns out that every process executes at least one ϵ-move, i.e. $e_i \in alph(t)$ for every $i \in Proc$. So, we can decompose t as $u_0 e_{i_1} u_1 e_{i_2} \dots e_{i_n} u_n$ where $(i_1, \dots, i_n)$ is a permutation of $(1, \dots, n)$ and $e_{i_j} \notin alph(u_k)$ for $k < j$.
By Remark (b), either $Dom(a) \subseteq \{i_1, \dots, i_k\}$ or $Dom(a) \subseteq \{i_{k+1}, \dots, i_n\}$, for every letter $a \in alph(u_k)$, $k = 1, \dots, n$. Then, $u_k = u'_k u''_k$, where $Dom(u'_k) \subseteq \{i_{k+1}, \dots, i_n\}$ and $Dom(u''_k) \subseteq \{i_1, \dots, i_k\}$. It is easy to verify that $t = u'_0 \dots u'_n [e_1 \dots e_n] u''_0 \dots u''_n$.

Let be $t' = u'_0 \dots u'_n$ and $t'' = u''_0 \dots u''_n$. We now briefly prove that $t' \in T(M)$ and $t'' \in T(M')$. By definition of M', we can find $b \in \{0, 1\}$ such that:

$((F_1, b), \dots, (F_n, b)) \in \Delta'(((I_1, 0), \dots, (I_n, 0)), t'[e_1 \dots e_n]t'')$.

More precisely, observing the definition of δ'_{e_i}, we obtain:

$((F_1, 0), \dots, (F_n, 0)) \in \Delta'(((I_1, 0), \dots, (I_n, 0)), t')$;
$((I_1, 1), \dots, (I_n, 1)) \in \Delta'(((F_1, 0), \dots, (F_n, 0)), [e_1 \dots e_n])$;
$((F_1, b), \dots, (F_n, b)) \in \Delta'(((I_1, 1), \dots, (I_n, 1)), t'')$.

Since $e_i \notin alph(t')$, $i = 1, \dots, n$, it turns out that $\boldsymbol{F} \in \Delta(\boldsymbol{I}, t')$ and then $t' \in T(M)$. Furthermore, from these relationships, we can easily obtain:

$((F_1, 1 \Leftrightarrow b), \dots, (F_n, 1 \Leftrightarrow b)) \in \Delta'(((I_1, 0), \dots, (I_n, 0)), t'')$,

that implies $t'' \in T(M')$.

At this point, iterating the previous proof on the trace t'' it turns out

that t is of the form $t_1[e_1 \dots e_n]t_2 \dots t_{m-1}[e_1 \dots e_n]t_m$, with $t_j \in T(M)$, $j = 1, \dots, m$. ∎

Applying Theorem 4.2 to the automaton M', we can obtain a NAA accepting the language $T(M)^+$.

Now, we briefly discuss how it is possible to generalize the previous construction when there are no restrictions on the cardinality of sets $\boldsymbol{I}$ and $\boldsymbol{F}$.

The idea is very similar to particular case above considered. When a process P_i reaches a component of a final state, it can change its state, using an ϵ–move, to go in a component of an initial state.

The problem is to assure that all processes chose the same final and the same initial global state. This problem is solved in this way. At the start of the computation all processes are synchronized and an initial state $\boldsymbol{s} \in I$ and a final state $\boldsymbol{p} \in F$ are chosen. Then, the computation starts and every process simulates the corresponding process of M, keeping the pair $(\boldsymbol{s}, \boldsymbol{p})$ in its local state.

When the process P_i arrives to simulate the local state p_i of M, it can perform an empty transition choosing a new pair $(\boldsymbol{s'}, \boldsymbol{p'}) \in I \times F$.

When a symbol $a \in A$ is executed, all processes in $Dom(a)$ are synchronized and they compare the pairs of initial and final states contained in their proper memories. If these pairs do not coincide then none transition is possible. Since all traces in the language accepted by M are connected and since the automaton M records the letters of traces, this method assures that in an accepting computation of M' all kth empty transitions of processes correspond to the same initial and to the same final state. Formally, the automaton M is defined as follows:

- $S'_i = S_i \times I \times F \times \{0,1\}$, $i = 1, \dots, n$;
- $I' = \{((s_1, \boldsymbol{s}, \boldsymbol{p}, 0), \dots, (s_n, \boldsymbol{s}, \boldsymbol{p}, 0)) \mid \boldsymbol{s} \in I, \boldsymbol{p} \in F\}$;
- for $a \in A$ with $Dom(a) = \{i_1, \dots, i_k\}$, $s_j \in S_{i_j}$, $j = 1, \dots, k$, $\boldsymbol{q}_1, \dots, \boldsymbol{q}_k \in I$, $\boldsymbol{p}_1, \dots, \boldsymbol{p}_k \in F$, $b_1, \dots, b_k \in \{0,1\}$:

  $\delta'_a((s_1, \boldsymbol{q}_1, \boldsymbol{p}_1, b_1), \dots, (s_k, \boldsymbol{q}_k, \boldsymbol{p}_k, b_k)) =$
  $= \{((u_1, \boldsymbol{q}, \boldsymbol{p}, b), \dots, (u_k, \boldsymbol{q}, \boldsymbol{p}, b)) \mid (u_1, \dots, u_k) \in \delta_a(s_1, \dots, s_k)\}$,

if there are $\boldsymbol{q} \in I$, $\boldsymbol{p} \in F$, $b \in \{0,1\}$, s.t. $\boldsymbol{q}_1 = \ldots = \boldsymbol{q}_k = \boldsymbol{q}$, $\boldsymbol{p}_1 = \ldots = \boldsymbol{p}_k = \boldsymbol{p}$, and $b_1 = \ldots = b_k = b$;

$$\delta'_a((s_1, \boldsymbol{q}_1, \boldsymbol{p}_1, b_1), \ldots, (s_k, \boldsymbol{q}_k, \boldsymbol{p}_k, b_k)) = \emptyset,$$

otherwise.

- for $i = 1, \ldots, n$, $s_i \in S_i$, $\boldsymbol{q} \in I$, $\boldsymbol{p} \in F$, $b \in \{0,1\}$:

$$\delta'_{e_i}(s_i, \boldsymbol{q}, \boldsymbol{p}, b) = \begin{cases} \{(q'_i, \boldsymbol{q}', \boldsymbol{p}', 1 \Leftrightarrow b) \mid \boldsymbol{q}' \in I, \boldsymbol{p}' \in F\} & \text{if } s_i = p_i \\ \emptyset & \text{otherwise;} \end{cases}$$

- $F' = \{((p_1, \boldsymbol{q}, \boldsymbol{p}, b), \ldots, (p_n, \boldsymbol{q}, \boldsymbol{p}, b)) \mid \boldsymbol{q} \in I, \boldsymbol{p} \in F, b \in \{0,1\}\}$.

Previous remarks (a), (b) and (c) hold also in this case. The correctness proof given in Theorem 5.3 can be easily generalized.

6 Conclusion

We conclude the paper with some considerations on the complexity of asynchronous automata obtained applying our constructions.

It is not difficult to see that all constructions given in the paper (union, product, restriction, decomposition and restricted iteration) produce NAAs with a polynomial number of states with respect to the number of states of the NAAs given as input.

Moreover, it is possible to observe that for a fixed concurrent alphabet (A, θ) the coiteration T^{co-*} of a trace language T can be denoted as explained in Section 5.3 using an expression involving operations of union, product, decomposition, restriction and restricted iteration, whose length is polynomial in the length of the recognizable expression given for T. Then, using the algorithms presented in this paper, for every recognizable expression it is possible to build a nondeterministic asynchronous automaton recognizing the language denoted by it and whose number of (global and local) states is polynomial in the length of the given expression.

References

[BBMS81] A. Bertoni, M. Brambilla, G. Mauri, and N. Sabadini. An application of the theory of free partially commutative monoids: asymptotic densities of trace languages. In *Proc. 10th MFCS*, Lecture Notes in Computer Science 118, pages 205–215, 1981.

[CM88] R. Cori and Y. Métivier. Approximation of a trace, asynchronous automata and the ordering of events in a distributed system. In *Proc. 15th ICALP*, Lecture Notes in Computer Science 317, pages 147–161, 1988.

[HU79] J. Hopcroft and J. Ullman. *Introduction to automata theory, languages and computations.* Addison–Wesley, 1979.

[Maz77] A. Mazurkiewicz. Concurrent program schemes and their interpretations. Technical Report DAIMI Rep. PB–78, Aarhus University, 1977.

[Och85] E. Ochmański. Regular behaviour of concurrent systems. *EATCS Bullettin*, 27:56–67, 1985.

[Zie87] W. Zielonka. Notes on finite asynchronous automata. *RAIRO Inf. Theor.*, 21:99–135, 1987.

Automata with Storage on Infinite Words

(Extended Abstract) [1]

Joost Engelfriet and Hendrik Jan Hoogeboom

Leiden University, Department of Computer Science
P.O. Box 9512, 2300 RA Leiden, The Netherlands

Abstract. For any storage type X, the ω-languages accepted by X-automata are investigated. Six accepting conditions (including those introduced by Landweber) are compared for X-automata. The inclusions between the corresponding six families of ω-languages are essentially the same as for finite state automata. Apart from unrestricted automata also real-time and deterministic automata are considered. The main tools for this investigation are (1) a characterization of the ω-languages accepted by X-automata in terms of (inverse) X-transductions of finite state ω-languages, and (2) the existence of topological upper bounds on some of the families of accepted ω-languages (independent of the storage type X).

Introduction

An automaton $\mathcal{A}$ that is meant to work on finite input words may as well be given an infinite input word u: it works on u as if u were a 'very

[1] This paper was previously presented at ICALP'89 and appeared in the proceedings of the conference ([EngHoo89]). A full version of the paper is scheduled to be published in Theoretical Computer Science in the spring of 1993.

large' finite word. The essential difference is in the way that $\mathcal{A}$ accepts u; obviously one cannot use acceptance by final state as for finite words.

The first one to use automata to accept infinite words (with a particular acceptance criterion) was Büchi (in solving a decision problem in logic, [Büc60]). Another criterion was given by Muller ([Mul63]).*A* deterministic finite state automaton $\mathcal{A}$ accepts an infinite word u in the fashion of Muller if the set of states entered by $\mathcal{A}$ infinitely often during its computation on u belongs to a given family of 'final' state sets. This *family* replaces the usual *set* of final states. Five criteria for accepting infinite words were proposed by Landweber in [Lan69], including those introduced by Büchi and Muller, and he characterized the five corresponding families of infinitary languages accepted by deterministic finite state automata in a topological setting.

The relative power of these five acceptance criteria was subsequently compared for (nondeterministic) finite state automata ([Hos72, StaWag74]), pushdown automata ([Lin76, CohGol77, CohGol78a]), Turing machines ([WagSta77, CohGol78b]) and Petri nets ([Val83]). If one compares the results of these investigations, one notices some striking similarities (see the survey [Sta87]). It seems that the acceptance types have the same relative power independently of the storage used by the automaton involved. This observation is the main motivation for the present paper. Using a general framework we want to explain the similarities between the results obtained for the various specific types of automata (as is done for automata on finite words in [Gin75]). Our abstract model of storage is called a *storage type*. It describes the storage configurations together with the tests and transformations that can be applied to them. Automata equipped with a specific storage X (and a one-way input-tape) are called X-automata. We study six (rather than five) families of ω-languages that can be accepted by an X-automaton using six different acceptance criteria on the sequence of states entered by the automaton during a computation. (It should be noted that acceptance can also be defined in terms of the storage configurations rather than the states, see [Sta77], but this will give quite different results, cf. [Val83]). A possible approach to comparing the six acceptance criteria is by giving constructions on automata that show how one acceptance

type can be simulated by another. In fact, as observed in [CohGol77, Val83], it is not too difficult to generalize most of the constructions given in [Hos72] for finite state automata, simply by 'adding' storage instructions to the transitions. Hence it is not much of a surprise that the inclusions between the six families for X-automata are similar to those formed by the families for finite state automata. Of course, this is a rather boring and time-consuming approach. Also, if one wants to study X-automata satisfying a particular property (as, e.g., being real-time or deterministic), it is necessary to check for each of the constructions whether it preserves the property under consideration (and if not, to adapt the construction). We use a more efficient way of transferring the results for finite state automata to arbitrary storages. Our main tool is a characterization of the ω-languages accepted by X-automata in terms of (infinitary) transductions applied to the ω-languages accepted by finite state automata. Since we do not use the acceptance criteria to define transductions, this single result can be used to show that the inclusions that hold between the six families of finite state ω-languages are also valid for X-automata. This of course does not indicate whether or not an inclusion is strict. We show that the topological upper-bounds on the complexity of accepted languages as given by Landweber for deterministic finite state automata can be generalized to X-automata (as already suggested in [Lan69]). This implies that for deterministic X-automata the inclusions are always strict (and the same holds for real-time automata).

In Section 3 we study both arbitrary and deterministic X-automata. First we present the above-mentioned characterization of the corresponding families of ω-languages (Theorem 3.3). From this we obtain the hierarchy for ω-languages accepted by X-automata (Theorem 3.5). For specific storage types the hierarchy can be strict or it can collapse into a single family. We give a sufficient condition for such a collapse (Theorem 3.7). Real-time automata are investigated in Section 4. The inclusions between the families of ω-languages accepted by real-time automata are very similar to those found in Section 3 (see Theorem 4.4). Here however, the inclusions are always strict. The counter-examples are obtained by establishing topological upper-bounds that are independent of the stor-

age type. We return to deterministic automata in Section 5. Again we obtain topological upper-bounds on the accepted ω-languages. Together with our basic characterization (given in Section 3) this is used to establish a proper hierarchy similar to the hierarchy for deterministic finite state automata (Theorem 5.4). In the final section we study a storage type of 'maximal power'. The families of ω-languages accepted by automata of this type belong to the lower levels of the topological hierarchy of Borel sets (Theorems 6.3 and 6.6). These results are similar to those obtained in [Arn83] and [Sta84] for transition systems.

1. Preliminaries

We assume the reader to be familiar with the basic notions of infinitary languages, e.g., as discussed in one of the following surveys and introductions: [Eil74, HooRoz86, Sta87, Tho88].

The symbol $\subseteq$ ($\subset$) denotes set inclusion (strict set inclusion, respectively); in diagrams we will use $\Rightarrow$ (and $\rightarrow$, respectively). We use $\sqcap$ to indicate non-disjointness of sets, i.e., $U \sqcap V$ if $U \cap V \neq \emptyset$.

As usual, Σ^* and Σ^ω denote the sets of finite words and infinite words (or ω-words) over the alphabet Σ. Their subsets are called (finitary) languages and infinitary languages (or ω-languages), respectively. The empty word is denoted by Λ. $u[n]$ denotes the prefix of length n of a (ω-)word u (when defined), and $pref(\mathrm{v})$ denotes the set of (finite) prefixes of v. For a (ω-)language K, $pref(K) = \cup\{pref(u) | u \in K\}$. An infinite sequence of finite words $\langle x_i \rangle_{i \in \mathbf{N}}$ such that each x_i is a prefix of x_{i+1} defines a unique element u of $\Sigma^* \cup \Sigma^\omega$ by taking the 'least upper bound' of the sequence, i.e., the shortest u that has each x_i as a prefix. u will be denoted by $lub\langle x_i \rangle_{i \in \mathbf{N}}$.

Let $K \subseteq \Sigma^*$ be a finitary language. The *ω-power* of K, denoted by K^ω, is $\{u \in \Sigma^\omega | u = lub\langle x_i \rangle_{i \in \mathbf{N}}$, where $x_0 \in K$ and $x_{i+1} \in x_i \cdot K$ for $i \in \mathbf{N}\}$, the *adherence* of K, denoted $adh(K)$, is $\{u \in \Sigma^\omega | pref(u) \subseteq pref(K)\}$, and the *limit* of K, denoted $lim(K)$, is $\{u \in \Sigma^\omega | pref(u) \cap K$ is infinite$\}$.

Σ^ω can be turned into a (compact) metric space by defining the distance function $d(u,v) = 2^{-\min\{n | u[n] \neq v[n]\}}$ for $u \neq v$. We will use $\mathcal{G}$, and $\mathcal{F}$ to denote the family of open, respectively closed, sets in this topological space. Furthermore, $\mathcal{G}_\delta$ is the family of denumerable intersections of open sets, and $\mathcal{F}_\sigma$ is the family of denumerable unions of closed sets. There is a close correspondence between the ω-languages in these families (that form the lower levels of the *Borel hierarchy*) and the language theoretical operations given above (see, e.g., [Lan69, StaWag74, BoaNiv80]).

1.1. Proposition. Let $L \subseteq \Sigma^\omega$. Then
(1) $L \in \mathcal{G}$ if and only if $L = K \cdot \Sigma^\omega$ for some $K \subseteq \Sigma^*$,
(2) $L \in \mathcal{F}$ if and only if $L = adh(K)$ for some $K \subseteq \Sigma^*$, and
(3) $L \in \mathcal{G}_\delta$ if and only if $L = lim(K)$ for some $K \subseteq \Sigma^*$. ■

1.2. Proposition. $0^*1 \cdot \{0,1\}^\omega \in \mathcal{G} \Leftrightarrow \mathcal{F}$, $\{0,1\}^* \cdot 1^\omega \in \mathcal{F}_\sigma \Leftrightarrow \mathcal{G}_\delta$, and $(0^*1)^\omega \in \mathcal{G}_\delta \Leftrightarrow \mathcal{F}_\sigma$. ■

2. Automata on ω-Words: Definitions

For finite words, the general notion of an automaton, using some kind of storage, was introduced in [HopUll67, Sco67, GinGre69]. The resulting AFA theory (Abstract Families of Automata) provides a useful framework for a uniform investigation of different types of automata (see [Gin75]). Here we attempt to set up a similar theory for automata on infinite words (see also [Sta77]). The particular variation of AFA theory that we use here is similar to the one in [EngVog87].

2.1. Storage and Automata

A *storage type* is a 5-tuple $\mathrm{X} = (C, C_{in}, P, F, \mu)$, where C is a set of (*storage*) *configurations*, $C_{in} \subseteq C$ is a set of *initial* (*storage*) *configura-*

tions, P is a set of *predicate symbols*, F is a set of *instruction symbols*, $P \cap F = \emptyset$, and μ is a *meaning function* which assigns to each $p \in P$ a (total) mapping $\mu(p) : C \to \{true, false\}$, and to each $f \in F$ a partial function $\mu(f) : C \to C$. The set of all Boolean expressions over P, using the Boolean connectives $\wedge$, $\vee$ and $\neg$, and the constants *true* and *false*, is denoted by $\mathrm{BE}(P)$; elements of this set are called *tests*. The meaning function is extended to $\mathrm{BE}(P)$ in the obvious way. We extend μ also from F to F^* by defining $\mu(\Lambda)$ to be the identity on C and by setting $\mu(f\phi) = \mu(\phi) \circ \mu(f)$ for $\phi \in F^*$ and $f \in F$.

2.1. Example. The storage type *push-down*, denoted PD, is defined by PD $= (C, C_{in}, P, F, \mu)$, where $C =$, $^+$, for a fixed infinite set , (of push-down symbols), $C_{in} =$, , $P = \{top = \gamma | \gamma \in$, $\} \cup \{bottom\}$, $F = \{push(\gamma) | \gamma \in$, $\} \cup \{pop\}$, and, for $c = au$ with $a \in$, and $u \in$, *, $\mu(top = \gamma)(c) = true$ iff $\gamma = a$, $\mu(bottom)(c) = true$ iff $u = \Lambda$, $\mu(push(\gamma))(c) = \gamma c$, $\mu(pop)(c) = u$ if $u \neq \Lambda$, and undefined otherwise.

The storage type *counter* is CTR $= (\mathbf{N}, \{0\}, \{zero\}, \{incr, decr\}, \mu)$, where, for $n \in \mathbf{N}$, $\mu(zero)(n) = true$ iff $n = 0$, $\mu(incr)(n) = n + 1$, and $\mu(decr)(n) = n \Leftrightarrow 1$ if $n \geq 1$, and undefined if $n = 0$. ∎

In the rest of this paper let X $= (C, C_{in}, P, F, \mu)$ be an arbitrary storage type.

An X-*transducer* is a construct $\mathcal{A} = (Q, \Sigma, \delta, q_{in}, c_{in}, \Delta)$, where Q is the finite set of *states*, $\Sigma(\Delta)$ is the *input* (*output*) *alphabet*, $q_{in} \in Q$ is the *initial state*, $c_{in} \in C_{in}$ is the *initial storage configuration*, and the *finite control* δ is a finite subset of $Q \times (\Sigma \cup \{\Lambda\}) \times \mathrm{BE}(P) \times Q \times F^* \times \Delta^*$, the elements of which are called *transitions*.
We say that $\mathcal{A}$ is *real-time* if $\delta \subseteq Q \times \Sigma \times \mathrm{BE}(P) \times Q \times F^* \times \Delta^*$.
$\mathcal{A}$ is *deterministic* if, for every two different transitions $(q_i, a_i, \beta_i, q_i', \phi_i, w_i)$, $i = 1, 2$, from δ with $q_1 = q_2$, either $a_1 \neq a_2$ and $a_1, a_2 \neq \Lambda$ or $\mu(\beta_1 \wedge \beta_2)(c) = false$ for every $c \in C$.

An *instantaneous description* (*ID*) *of* $\mathcal{A}$ is an element (q, x, c, y) of $Q \times \Sigma^* \times C \times \Delta^*$; it intuitively means that $\mathcal{A}$ is in state q, has read x from the input tape, has c as its storage configuration, and has written y on its output tape. The *step relation* $\vdash_{\mathcal{A}}$ *of* $\mathcal{A}$, is the binary relation

on $Q \times \Sigma^* \times C \times \Delta^*$ defined by $(q, x, c, y) \vdash_{\mathcal{A}} (q', x', c', y')$ if there exists a transition $(q, a, \beta, q', \phi, w) \in \delta$ such that $\mu(\beta)(c) = true$, $c' = \mu(\phi)(c)$, $x' = xa$, and $y' = yw$. Intuitively this means that if $\mathcal{A}$ is in state q and has the storage configuration c, it may use the transition $(q, a, \beta, q', \phi, w)$ provided c satisfies the test β, and then it changes its state to q', reads a from its input tape, performs ϕ to the storage configuration, and writes w on its output tape.

A (*infinite*) *run of* $\mathcal{A}$ is an infinite sequence $r = \langle \tau_i \rangle_{i \in \mathbf{N}}$ of ID's such that $\tau_0 = (q_{in}, \Lambda, c_{in}, \Lambda)$, and $\tau_i \vdash_{\mathcal{A}} \tau_{i+1}$ for each $i \in \mathbf{N}$; it is a run *on input* $lub\langle x_i \rangle_{i \in \mathbf{N}}$, and *with output* $lub\langle y_i \rangle_{i \in \mathbf{N}}$ where $\tau_i = (q_i, x_i, c_i, y_i)$. The sequence $\langle q_i \rangle_{i \in \mathbf{N}}$ is called the *state sequence of* the run r.

The (*infinitary*) *transduction* of $\mathcal{A}$, denoted $T(\mathcal{A})$, is defined as $\{(u, v) \in \Sigma^\omega \times \Delta^\omega \mid$ there is a run of $\mathcal{A}$ on input u with output $v\}$. If $\mathcal{A}$ has no run on an infinite input word with a finite output word, then $\mathcal{A}$ is called ω-*preserving*. XT (XT_ω) denotes the family of infinitary transductions of (ω-preserving) X-transducers. The corresponding families of transductions of deterministic or real-time X-transducers are denoted by d-XT (d-XT_ω) and r-XT (r-XT_ω), respectively.

2.2. (σ,ρ)-Accepting Infinite Runs

We will now discuss how an X-transducer $\mathcal{A}$ may be used to accept ω-languages. Since, in this case, we are not interested in $\mathcal{A}$'s output, $\mathcal{A}$ is called an X-*automaton*. We drop the output component from $\mathcal{A}$, and from its transitions and ID's.

Let Q be a set and let f be a mapping $\mathbf{N} \to Q$ (i.e., an infinite sequence $\langle f(i) \rangle_{i \in \mathbf{N}}$ over Q). The *range of* f is the set $ran(f) = \{q \in Q \mid f(i) = q$ for some $i \in \mathbf{N}\}$; the *infinity set of* f is the set $in(f) = \{q \in Q \mid f(i) = q$ for infinitely many $i \in \mathbf{N}\}$.

Let $\mathcal{D} \subseteq 2^Q$ be a family of subsets of Q. Let ρ be one of the relations $\subseteq$, $\sqcap$ or $=$, and let σ be either *ran* or *in*. We say that an infinite sequence $f : \mathbf{N} \to Q$ is (σ, ρ)-*accepting with respect to* $\mathcal{D}$ if there exists a set $D \in \mathcal{D}$ such that $\sigma(f) \rho D$.

The relation between the notation we use (see [StaWag74]) and the

five types of 'i-acceptance' as originally defined in [Lan69] are given in the following table, together with a short intuitive name for some of these types of acceptance. $(ran, =)$-acceptance, not considered by Landweber, was first studied in [StaWag74].

$(ran, \sqcap)$	1-accepting	at least once	
$(ran, \subseteq)$	$1'$-accepting	always	
$(ran, =)$	—		
$(in, \sqcap)$	2-accepting	infinitely often	(Büchi)
$(in, \subseteq)$	$2'$-accepting	from some moment on	
$(in, =)$	3-accepting		(Muller)

Let $\mathcal{A} = (Q, \Sigma, \delta, q_{in}, c_{in})$ be an X-automaton, and let $\mathcal{D} \subseteq 2^Q$ be a family of subsets of Q. A run of $\mathcal{A}$ is called (σ, ρ)-*accepting with respect to* $\mathcal{D}$ if its state sequence is (σ, ρ)-accepting with respect to $\mathcal{D}$.

The ω-*language* (σ, ρ)-*accepted by* $\mathcal{A}$ *with respect to* $\mathcal{D}$, denoted by $L_{\sigma,\rho}(\mathcal{A}, \mathcal{D})$, is the set $\{u \in \Sigma^\omega |$ there is a run of $\mathcal{A}$ on input u that is (σ, ρ)-accepting with respect to $\mathcal{D}\}$. The family of ω-languages (σ, ρ)-accepted by X-automata (with respect to some family of state sets) is denoted by $\mathrm{XL}_{\sigma,\rho}$. Again we use the prefixes d- or r- if we restrict our considerations to deterministic or real-time automata. Finally, for a set D of states of $\mathcal{A}$, the (*finitary*) *language* $L_*(\mathcal{A}, D)$ *accepted by* $\mathcal{A}$ *with respect to* D is the set $\{x \in \Sigma^* | (q_{in}, \Lambda, c_{in}) \vdash^*_{\mathcal{A}} (q, x, c)$ for some $q \in D$ and $c \in C\}$, where $\vdash^*_{\mathcal{A}}$ is the reflexive and transitive closure of $\vdash_{\mathcal{A}}$.

In the literature it is sometimes required that $\mathcal{A}$ is 'total', e.g., in the sense that $\mathcal{A}$ has a run on every input word. We will not consider totality. Requiring totality changes (in general) the class $\mathrm{XL}_{\sigma,\rho}$.

A special storage type is used to model finite state automata. The *trivial storage type* FS equals $(\{c0\}, \{c0\}, \emptyset, \emptyset, \emptyset)$ for some arbitrary object $c0$. Since $\emptyset^* = \{\Lambda\}$ the transitions of an FS-automaton can be assumed to be of the form $(q, a, true, q', \Lambda)$.

2.2. Example. Let $\mathcal{A}$ be the (deterministic and real-time) FS-automaton

with state set $Q = \{q_0, q_1\}$, input alphabet $\Sigma = \{0,1\}$, initial state q_0, and transitions $(q_i, j, true, q_j, \Lambda)$ for $i, j \in \{0,1\}$. Let $\mathcal{D} = \{\{q_1\}\}$, and $\mathcal{Q} = \{Q\}$. Then

$$L_{ran,\sqcap}(\mathcal{A},\mathcal{D}) = 0^*1 \cdot \{0,1\}^\omega \qquad L_{ran,\sqcap}(\mathcal{A},\mathcal{Q}) = L_{ran,\subseteq}(\mathcal{A},\mathcal{Q}) = \{0,1\}^\omega,$$
$$L_{ran,\subseteq}(\mathcal{A},\mathcal{D}) = L_{ran,=}(\mathcal{A},\mathcal{D}) = \emptyset, \qquad L_{ran,=}(\mathcal{A},\mathcal{Q}) = 0^*1 \cdot \{0,1\}^\omega,$$
$$L_{in,\sqcap}(\mathcal{A},\mathcal{D}) = (0^*1)^\omega, \qquad L_{in,\sqcap}(\mathcal{A},\mathcal{Q}) = L_{in,\subseteq}(\mathcal{A},\mathcal{Q}) = \{0,1\}^\omega,$$
$$L_{in,\subseteq}(\mathcal{A},\mathcal{D}) = L_{in,=}(\mathcal{A},\mathcal{D}) = \{0,1\}^* \cdot 1^\omega, \qquad L_{in,=}(\mathcal{A},\mathcal{Q}) = (0^*11^*0)^\omega.$$

■

It is customary in the literature to define regular ω-languages using *real-time* finite state automata. It can be shown that this restriction does not influence the families of (σ,ρ)-accepted finite state ω-languages, i.e., $\mathrm{FSL}_{\sigma,\rho} = \text{r-FSL}_{\sigma,\rho}$ and $\text{d-FSL}_{\sigma,\rho} = \text{dr-FSL}_{\sigma,\rho}$. Hence we have the following well-known relationships for these families (see, e.g., [Wag79] for references). Note that $\to$ denotes proper inclusion.

2.3. Proposition.

$$\begin{array}{ccccccccc}
 & & & & & & & & \mathrm{FSL}_{in,\sqcap} \\
 & & & & & & & & \| \\
\mathrm{FSL}_{ran,\subseteq} & & \mathrm{FSL}_{ran,\sqcap} & = & \mathrm{FSL}_{ran,=} & = & \mathrm{FSL}_{in,\subseteq} & & \mathrm{FSL}_{in,=} \\
\| & & & & & & \| & & \| \\
\| & & & & & & \text{d-FSL}_{in,\subseteq} & & \| \\
\text{d-FSL}_{ran,\subseteq} & \to & \text{d-FSL}_{ran,\sqcap} & \to & \text{d-FSL}_{ran,=} & \nearrow\searrow & & \searrow\nearrow & \text{d-FSL}_{in,=} \\
 & & & & & & \text{d-FSL}_{in,\sqcap} & &
\end{array}$$

■

Note that when requiring totality the diagram is slightly different, caused by the fact that $\mathrm{FSL}_{ran,\sqcap}$ and $\text{d-FSL}_{ran,\sqcap}$ become smaller.

Finally we need (but do not define formally) the notion of the *product*

of two *storage types* X_1 and X_2, denoted by $X_1 \times X_2$. It combines the power of the two storages X_1 and X_2 working in an independent fashion. Thus, e.g., the CTR×CTR-automaton is the well-known two-counter automaton. The product of n storage types, all equal to X, is denoted by X^n. An X^*-automaton is an X^n-automaton for some $n \in \mathbf{N}$. Thus, e.g., each CTR^*-automaton has an arbitrary (but fixed) number of counters.

3. The Basic Characterization

In the next lemma we show how to separate an X-automaton in two phases: a phase in which the input is processed, and an acceptance phase. The first phase can be realized by an X-transducer (without acceptance criterion), and the second phase by a finite state automaton.

3.1. Lemma. $\mathrm{XL}_{\sigma,\rho} \subseteq \mathrm{XT}_\omega^{-1}(\text{d-}\mathrm{FSL}_{\sigma,\rho})$ and
$\text{d-}\mathrm{XL}_{\sigma,\rho} \subseteq \text{d-}\mathrm{XT}_\omega^{-1}(\text{d-}\mathrm{FSL}_{\sigma,\rho})$.

Proof. Let $\mathcal{A}$ be an X-automaton with input alphabet Σ and state set Q; let $\mathcal{D}$ be a family of state sets for $\mathcal{A}$. Consider the infinitary language $K(\sigma,\rho)$ consisting of the ω-words over Q that are (σ,ρ)-accepting sequences wrt. $\mathcal{D}$. It belongs to $\text{d-}\mathrm{FSL}_{\sigma,\rho}$ since it is (σ,ρ)-accepted by the deterministic finite state automaton $\mathcal{B} = (Q \cup \{q^o\}, Q, \delta_1, q^o, c0)$ with $q^o \notin Q$ and $\delta_1 = \{(q', q, true, q, \Lambda) | q' \in Q \cup \{q^o\}, q \in Q\}$. One easily sees that for $r \in Q^\omega$, the run r_1 of $\mathcal{B}$ on r satisfies $ran(r_1) = ran(r) \cup \{q^o\}$ and $in(r_1) = in(r)$. Thus $K(\sigma,\rho) = L_{\sigma,\rho}(\mathcal{B}, \mathcal{D}^o)$ with $\mathcal{D}^o = \{D \cup \{q^o\} | D \in \mathcal{D}\}$ if (σ,ρ) equals $(ran, =)$ or $(ran, \subseteq)$, and $K(\sigma,\rho) = L_{\sigma,\rho}(\mathcal{B}, \mathcal{D})$ in the four remaining cases.

Modify $\mathcal{A}$ such that at each step it outputs its state, i.e., take $\Delta = Q$ and replace every transition (q, a, β, q', ϕ) by the transition $(q, a, \beta, q', \phi, q)$. This gives an ($\omega$-preserving) X-transducer $\mathcal{M}$ that maps each ω-word u over Σ onto the state sequences of the runs of $\mathcal{A}$ on u. u is (σ,ρ)-accepted by $\mathcal{A}$ if one of these state sequences belongs to $K(\sigma,\rho)$. Hence, $L_{\sigma,\rho}(\mathcal{A}, \mathcal{D}) = \{u \in \Sigma^\omega | (u, r) \in T(\mathcal{M}) \text{ for some } r \in K(\sigma,\rho)\} = T(\mathcal{M})^{-1}(K(\sigma,\rho))$. ■

3.2. Lemma. For storage types X_1 and X_2, $X_1 T_\omega^{-1}(X_2 L_{\sigma,\rho}) \subseteq (X_1 \times X_2) L_{\sigma,\rho}$ and d-$X_1 T_\omega^{-1}$(d-$X_2 L_{\sigma,\rho}$) $\subseteq$ d-$(X_1 \times X_2) L_{\sigma,\rho}$.

Proof. Let $\mathcal{M}$ be an ω-preserving X_1-transducer with input alphabet Σ, let $\mathcal{A}$ be an X_2-automaton with a family of state sets $\mathcal{D}$. The lemma is proved by constructing an $X_1 \times X_2$-automaton $\mathcal{B}$ that (σ,ρ)-accepts the ω-language $T(\mathcal{M})^{-1}(L_{\sigma,\rho}(\mathcal{A},\mathcal{D})) = \{u \in \Sigma_1^\omega \mid (u,v) \in T(\mathcal{M}) \text{ for some } v \in L_{\sigma,\rho}(\mathcal{A},\mathcal{D})\}$. Using a straightforward direct product construction we simulate $\mathcal{M}$ and $\mathcal{A}$ in two alternating phases: first we simulate $\mathcal{M}$ until it produces some nonempty output and we store this output in the states of $\mathcal{B}$, then we simulate $\mathcal{A}$ on this output. Using such a simulation, a run $r_\mathcal{B}$ of $\mathcal{B}$ on an ω-word u can be decomposed into a run of $\mathcal{M}$ on u and an infinite run $r_\mathcal{A}$ of $\mathcal{A}$ on the output v of $\mathcal{M}$. Here the fact that $\mathcal{M}$ is ω-preserving is essential in order to guarantee that v is infinite. If we restrict the state sequence of $r_\mathcal{B}$ to the component of the states corresponding to $\mathcal{A}$, it is equal (except for possible repetitions) to the state sequence of $r_\mathcal{A}$. This observation can be used to obtain a suitable family of state sets $\mathcal{D}'$ for $\mathcal{B}$. Some care has to be taken: $\mathcal{D}'$ depends on the acceptance type. ■

If we combine these lemma's we obtain our basic result: a characterization of the families $XL_{\sigma,\rho}$ and d-$XL_{\sigma,\rho}$ in terms of X-transductions and finite state ω-languages.

3.3. Theorem. $XL_{\sigma,\rho} = XT_\omega^{-1}(FSL_{\sigma,\rho})$ and
d-$XL_{\sigma,\rho}$ = d-XT_ω^{-1}(d-$FSL_{\sigma,\rho}$)

Proof. For $X_1 = X$ and $X_2 = FS$ we get $XL_{\sigma,\rho} \subseteq XT_\omega^{-1}(FSL_{\sigma,\rho}) \subseteq (X \times FS)L_{\sigma,\rho} \subseteq XL_{\sigma,\rho}$. ■

For some of the acceptance conditions the family $FSL_{\sigma,\rho}$ can be replaced by a single ω-language. Intuitively this ω-language models the acceptance condition. A similar result was obtained in [WagSta77] for ω-languages accepted by Turing machines, and (implicitly) in [LatTim86] for regular ω-languages.

3.4. Theorem. $\mathrm{XL}_{ran,\subseteq} = \mathrm{XT}_\omega^{-1}(\{1^\omega\})$, $\mathrm{XL}_{ran,\sqcap} = \mathrm{XT}_\omega^{-1}(\{0^*1 \cdot \{0,1\}^\omega\})$, $\mathrm{XL}_{in,\subseteq} = \mathrm{XT}_\omega^{-1}(\{\{0,1\}^*\cdot 1^\omega\})$, and $\mathrm{XL}_{in,\sqcap} = \mathrm{XT}_\omega^{-1}(\{(0^*1)^\omega\})$. Similar equalities hold for d-$\mathrm{XL}_{\sigma,\rho}$ and d-XT_ω^{-1}.

Proof. For $\rho \in \{\subseteq, \sqcap\}$ we may assume that $L \in \mathrm{XL}_{\sigma,\rho}$ is accepted by an X-automaton with respect to a family $\mathcal{D} = \{D\}$ (cf. [CohGol78a, Lemma 4.1.2]). Now (as in the proof of Lemma 3.1) change $\mathcal{A}$ into an X-transducer $\mathcal{M}$ that outputs 1 if its state is in D, and 0 otherwise. ■

Using Theorem 3.3, we can carry over the known inclusions for the families $\mathrm{FSL}_{\sigma,\rho}$ (see Proposition 2.3) directly to the families $\mathrm{XL}_{\sigma,\rho}$ for arbitrary X, without being forced to generalize the proofs for FS-automata. Thus we obtain our next main result.

3.5. Theorem. $\mathrm{XL}_{ran,\subseteq} \subseteq \mathrm{XL}_{ran,\sqcap} = \mathrm{XL}_{ran,=} = \mathrm{XL}_{in,\subseteq} \subseteq \mathrm{XL}_{in,\sqcap} = \mathrm{XL}_{in,=}$. ■

The corresponding result for deterministic automata will be presented in Section 5. We cannot conclude that the inclusions are strict in general like for finite state ω-languages (X = FS, Proposition 2.3) or push-down automata (X = PD, [CohGol77]). In fact, for certain storage types all six families $\mathrm{XL}_{\sigma,\rho}$ are equal (e.g., for Turing machines, see [CohGol78b]). In the remainder of this section we give a sufficient condition on X for all six families to be equal. It is based on the inclusion $\mathrm{FSL}_{in,\sqcap} \subseteq \mathrm{PDL}_{ran,\subseteq}$, which was proved in [CohGol77] using the pushdown as a counter.

The storage type *blind counter*, denoted by BC, is equal to the storage type counter (see Example 2.1), except that it has no predicate symbols (cf. [Gre78] where it is called a *partially* blind counter).

3.6. Lemma. $\mathrm{XL}_{in,\sqcap} \subseteq (\mathrm{X} \times \mathrm{BC})\mathrm{L}_{ran,\subseteq}$.

Proof. By Theorem 3.4, $\mathrm{XL}_{in,\sqcap} = \mathrm{XT}_\omega^{-1}(\{(0^*1)^\omega\})$, while $\mathrm{XT}_\omega^{-1}(\mathrm{BCL}_{ran,\subseteq}) \subseteq (\mathrm{X} \times \mathrm{BC})\mathrm{L}_{ran,\subseteq}$, by Lemma 3.2. Hence it suffices to show that $(0^*1)^\omega \in \mathrm{BCL}_{ran,\subseteq}$. We construct a BC-automaton $\mathcal{A}$ that uses its (blind) counter to ensure that during its runs it can read any fi-

nite number of consecutive 0's, but not infinitely many consecutive 0's. $\mathcal{A}$ has two states q_0 and q_1 and, for $i = 0, 1$, transitions $(q_i, 0, true, q_0, decr)$ and $(q_i, 1, true, q_1, \Lambda)$, and the transition $(q_1, \Lambda, true, q_1, incr)$. The initial state of $\mathcal{A}$ is q_1. Take $\mathcal{D} = \{\{q_0, q_1\}\}$. For each step on the letter 0 $\mathcal{A}$ decreases its counter. Whenever $\mathcal{A}$ reads the letter 1 it enters state q_1. In this state, before reading the next input letter, $\mathcal{A}$ guesses the number of 0's on the tape before the next 1, and increases its counter value by (at least) this amount. ■

Hence, if the storage type X can simulate an additional blind counter, all $\mathrm{XL}_{\sigma,\rho}$ are the same.

3.7. Theorem. If X can simulate X×BC, in the sense that $(\mathrm{X}\times\mathrm{BC})\mathrm{L}_{ran,\subseteq} \subseteq \mathrm{XL}_{ran,\subseteq}$, then $\mathrm{XL}_{ran,\subseteq} = \mathrm{XL}_{in,\sqcap}$. ■

Using this result we clearly reobtain the equality of the families of ω-languages (σ, ρ)-accepted by Turing machines ([CohGol78b]).

It is obvious that the storage type BC^* can simulate an additional blind counter. Hence for Petri net ω-languages, with acceptance with respect to bounded markings (cf. [Val83]), and Λ-labels allowed, the six families $\mathrm{BC}^*\mathrm{L}_{\sigma,\rho}$ are all equal. Although these families were not (explicitly) compared in the literature, the inclusion $\text{r-}\mathrm{BC}^*\mathrm{L}_{in,\sqcap} \subseteq \mathrm{BC}^*\mathrm{L}_{ran,\subseteq}$ is known (see [Car88, Theorem 3]).

4. Real-Time Automata

As in Section 3, we obtain a characterization of the families $\text{r-}\mathrm{XL}_{\sigma,\rho}$ in terms of X-transductions and finite state ω-languages. The proofs are slight variations of those of Lemma's 3.1 and 3.2.

4.1. Theorem. $\text{r-}\mathrm{XL}_{\sigma,\rho} = \text{r-}\mathrm{XT}_{\omega}^{-1}(\mathrm{FSL}_{\sigma,\rho})$. ■

This allows us, as in Section 3, to transfer the inclusions known for

the families $\mathrm{FSL}_{\sigma,\rho}$ directly to the families r-$\mathrm{XL}_{\sigma,\rho}$. Again we investigate when the remaining two inclusions are equalities and when they are strict. Perhaps somewhat surprisingly they turn out to be always strict, as shown next.

4.2. Lemma. Let $\mathcal{A}$ be a real-time X-automaton, with state set Q. Then $L_{ran,\subseteq}(\mathcal{A},\{Q\}) = adh(L_*(\mathcal{A},Q))$.

Proof. Let $L = L_*(\mathcal{A},Q)$. Note that $L_{ran,\subseteq}(\mathcal{A},\{Q\})$ is the set of all ω-words on which there exists a run of $\mathcal{A}$, without additional requirements on its state sequence. Assume that $u \in L_{ran,\subseteq}(\mathcal{A},\{Q\})$. This implies that $u[n] \in L$ for each $n \in \mathbf{N}$ and consequently $u \in adh(L)$. To prove the other inclusion let $u \in adh(L)$. Thus for each $n \in \mathbf{N}$ $\mathcal{A}$ accepts the prefix $u[n]$ of u. Define the sets E_n, $n \in \mathbf{N}$, of ID's of $\mathcal{A}$ that are reachable from the initial ID after reading $u[n]$. Since each of these sets is finite and nonempty, König's Lemma can be used to obtain a run of $\mathcal{A}$ on u. ■

4.3. Lemma. (1) r-$\mathrm{XL}_{ran,\subseteq} \subseteq \mathcal{F}$ and (2) r-$\mathrm{XL}_{in,\subseteq} \subseteq \mathcal{F}_\sigma$.

Proof. Since $\mathcal{F}$ and $\mathcal{F}_\sigma$ are closed under union, it suffices to consider ω-languages (σ,ρ)-accepted with respect to a single state set. Let $\mathcal{A} = (Q,\Sigma,\delta,q_{in},c_{in})$ be a real-time X-automaton and let $D \subseteq Q$.
$L_{ran,\subseteq}(\mathcal{A},\{D\})$ does not change by restricting $\mathcal{A}$ to the states from D. Hence (1) is a consequence of the lemma above.

Each $(in,\subseteq)$-accepting state sequence (w.r.t. $\{D\}$) of $\mathcal{A}$ falls apart into an initial part without restrictions, followed by an infinite part that stays within D. Thus

$$L_{in,\subseteq}(\mathcal{A},\{D\}) = \bigcup_{(q_{in},\Lambda,c_{in})\vdash^*(q,x,c)} L_{ran,\subseteq}(\mathcal{A}(q,x,c),\{D \cup Q_{q,x,c}\}),$$

where $\mathcal{A}(q,x,c)$ equals $\mathcal{A}$, except that it has a new set of transitions, leading from a new initial state q_{in} to q, reading x from the input, and transforming c_{in} into c — these transitions copy the computation $(q_{in},\Lambda,c_{in}) \vdash^*_{\mathcal{A}} (q,x,c)$. $Q_{q,x,c}$ is the set of states that were added to

obtain $\mathcal{A}(q,x,c)$. The above union is countable. Hence, by (1), every ω-language in r-XL$_{in,\subseteq}$ is a countable union of $\mathcal{F}$-sets, thus an $\mathcal{F}_\sigma$-set. ■

These topological upper-bounds yield examples to show that the remaining two inclusions are strict for every storage type. Thus we obtain the next main result.

4.4. Theorem. r-XL$_{ran,\subseteq}$ $\subset$ r-XL$_{ran,\sqcap}$ = r-XL$_{ran,=}$ = r-XL$_{in,\subseteq}$ $\subset$ r-XL$_{in,\sqcap}$ = r-XL$_{in,=}$.

Proof. According to Proposition 1.2, $0^*1 \cdot \{0,1\}^\omega \notin \mathcal{F} \supseteq$ r-XL$_{ran,\subseteq}$; however $0^*1 \cdot \{0,1\}^\omega \in$ r-XL$_{ran,\sqcap}$ (cf. Example 2.2). In the same way we have $(0^*1)^\omega \notin \mathcal{F}_\sigma \supseteq$ r-XL$_{in,\subseteq}$, whereas $(0^*1)^\omega \in$ r-XL$_{in,\sqcap}$. ■

Perhaps one of the main reasons for obtaining the strict inclusions above, is that König's Lemma is applicable to real-time automata (Lemma 4.2). This argumentation can be extended to a larger class of automata. An X-automaton $\mathcal{A}$ has *finite delay* if there is no infinite run of $\mathcal{A}$ on a finite word. All results for real-time automata proved in this section also hold for automata with finite delay. We use the prefix f- to indicate finite delay.

In general the families f-XL$_{\sigma,\rho}$ and r-XL$_{\sigma,\rho}$ are not equal. The ω-language BIN$_\omega$ = $\{x \cdot a^k | x \in \{0,1\}^*, 0 \leq k \leq nr(x)\} \cdot b^\omega \cup \{0,1\}^\omega$, where $nr(x)$ denotes the integer represented by $x \in \{0,1\}^*$ as a binary number, can be accepted by a BC^2-automaton with finite delay but not by a real-time BC*-automaton. More precisely, BIN$_\omega$ $\in$ f-BC^{2}L$_{ran,\subseteq}$ $\Leftrightarrow$r-BC*L$_{in,\sqcap}$. This example is essentially the one given by Jantzen ([Jan79]) to show that Λ-labelled Petri nets are more powerful than real-time Petri nets (when accepting finitary languages). It is perhaps interesting to note that Petri nets with finite delay were considered in the context of ω-languages. In [Car84] these nets are called *prompt* nets. This paper however focusses on nets that are 1-prompt, i.e., nets in which Λ-transitions are allowed, but not two consecutive Λ-transitions in a firing sequence (run). From our previous results it now follows that the following diagram holds for Petri nets (with acceptance with respect

to bounded places, see [Val83] for real-time nets). We conjecture that the indicated inclusion ($\Uparrow$) is an equality.

4.5 Theorem.

$$\begin{array}{ccccc}
\mathrm{BC^*L}_{ran,\subseteq} & = & \mathrm{BC^*L}_{ran,\sqcap} & = & \mathrm{BC^*L}_{in,\sqcap} \\
 & & & & \Uparrow \\
\mathrm{f} \Leftrightarrow \mathrm{BC^*L}_{ran,\subseteq} & \rightarrow & \mathrm{f} \Leftrightarrow \mathrm{BC^*L}_{ran,\sqcap} & \rightarrow & \mathrm{f} \Leftrightarrow \mathrm{BC^*L}_{in,\sqcap} \\
\uparrow & & \uparrow & & \uparrow \\
\mathrm{r} \Leftrightarrow \mathrm{BC^*L}_{ran,\subseteq} & \rightarrow & \mathrm{r} \Leftrightarrow \mathrm{BC^*L}_{ran,\sqcap} & \rightarrow & \mathrm{r} \Leftrightarrow \mathrm{BC^*L}_{in,\sqcap}
\end{array}$$

■

5. Deterministic Automata

In this section we return to deterministic automata. Again we show that the diagram for d-$\mathrm{FSL}_{\sigma,\rho}$ (Proposition 2.3) holds for arbitrary d-$\mathrm{XL}_{\sigma,\rho}$.

5.1. Lemma. Let $\mathcal{A}$ be a deterministic X-automaton with state set Q and input alphabet Σ, and let $\mathcal{D} \subseteq 2^Q$. Then
(1) $L_{ran,\subseteq}(\mathcal{A}, \{Q\})) = adh(L_*(\mathcal{A}, Q))$,
(2) $L_{ran,\sqcap}(\mathcal{A}, \mathcal{D}) = L_*(\mathcal{A}, \cup\mathcal{D}) \cdot \Sigma^\omega \cap adh(L_*(\mathcal{A}, Q))$, and
(3) $L_{in,\sqcap}(\mathcal{A}, \mathcal{D}) = lim(L_*(\mathcal{A}, \cup\mathcal{D}))$. ■

The above (simple) result enables us to give topological upper-bounds on the ω-languages that are accepted by deterministic automata. Given two families $\mathcal{K}$ and $\mathcal{L}$, $\mathcal{K} \wedge \mathcal{L}$ denotes $\{K \cap L | K \in \mathcal{K}, L \in \mathcal{L}\}$, and $\mathcal{B}(\mathcal{K})$ denotes the boolean closure of $\mathcal{K}$.

5.2. Corollary. (1) d-$\mathrm{XL}_{ran,\subseteq} \subseteq \mathcal{F}$,
(2) d-$\mathrm{XL}_{ran,\sqcap} \subseteq \mathcal{F} \wedge \mathcal{G}$,
(3) d-$\mathrm{XL}_{ran,=} \subseteq \mathcal{B}(\mathcal{F})$,
(4) d-$\mathrm{XL}_{in,\subseteq} \subseteq \mathcal{F}_\sigma$,
(5) d-$\mathrm{XL}_{in,\sqcap} \subseteq \mathcal{G}_\delta$, and
(6) d-$\mathrm{XL}_{in,=} \subseteq \mathcal{B}(\mathcal{F}_\sigma)$.

Proof. (1,2,5) These are clear from Lemma 5.1.
(4) Can be shown just as the inclusion r-XL$_{in,\subseteq} \subseteq \mathcal{F}_\sigma$ in Lemma 4.3.
(3,6) Consider an arbitrary deterministic X-automaton $\mathcal{A}$ and a family $\mathcal{D}$ of state sets for $\mathcal{A}$. Then $L_{\sigma,=}(\mathcal{A},\mathcal{D}) = \bigcup_{D\in\mathcal{D}}(L_{\sigma,\subseteq}(\mathcal{A},\{D\}) \Leftrightarrow L_{\sigma,\sqcap}(\mathcal{A},\{Q \Leftrightarrow D\}))$. Now d-XL$_{ran,=} \subseteq \mathcal{B}(\mathcal{F})$ and d-XL$_{in,=} \subseteq \mathcal{B}(\mathcal{F}_\sigma)$ follow. ■

5.3. Lemma. $\{0,1\}^\omega \Leftrightarrow 0^*1\cdot 0^\omega \notin \mathcal{F} \wedge \mathcal{G}$.

Proof. Assume that $K = \{0,1\}^\omega \Leftrightarrow 0^*1\cdot 0^\omega$ is of the form $L_1\cdot\{0,1\}^\omega \cap adh(L_2)$ for finitary languages L_1 and L_2. Since $0^\omega \in K \subseteq L_1\cdot\{0,1\}^\omega$, we have $0^n \in L_1$ for some $n \in \mathbf{N}$. On the other hand $0^n10^*1\cdot 0^\omega \subseteq K \subseteq adh(L_2)$, so $pref(0^n10^*) \subseteq pref(L_2)$. Consequently $0^n1\cdot 0^\omega \in L_1\cdot\{0,1\}^\omega \cap adh(L_2) = K$; a contradiction. ■

We now present the next main result.

5.4. Theorem. The following diagram holds:

$$\text{d-XL}_{ran,\subseteq} \;\to\; \text{d-XL}_{ran,\sqcap} \;\to\; \text{d-XL}_{ran,=} \;\begin{array}{c}\nearrow\\ \searrow\end{array}\; \begin{array}{c}\text{d-XL}_{in,\subseteq}\\ \\ \text{d-XL}_{in,\sqcap}\end{array} \;\begin{array}{c}\searrow\\ \nearrow\end{array}\; \text{d-XL}_{in,=}$$

Proof. The inclusions follow from Proposition 2.3 and Theorem 3.3. Using the topological upper-bounds we have obtained, we find (using Proposition 1.2 and Lemma 5.3) that $0^*1\cdot\{0,1\}^\omega \notin$ d-XL$_{ran,\subseteq}$, $\{0,1\}^\omega \Leftrightarrow 0^*1\cdot 0^\omega \notin$ d-XL$_{ran,\sqcap}$, $(0^*1)^\omega \notin$ d-XL$_{in,\subseteq}$, and $\{0,1\}^*\cdot 1^\omega \notin$ d-XL$_{in,\sqcap}$. However these ω-languages are in respectively d-XL$_{ran,\sqcap}$, d-XL$_{ran,=}$, d-XL$_{in,\sqcap}$, and d-XL$_{in,\subseteq}$, as the reader will easily verify. From these observations the strictness of the inclusions (and the incomparabilities) follow. ■

For some of the storages studied in the literature it was observed that the families d-XL$_{in,\subseteq}$ and d-XL$_{in,\sqcap}$ are 'complementary', i.e., one contains the complements of the ω-languages of the other. This is due to the fact that automata are often assumed to be 'total', i.e., they should

have a run on every possible input (this is called the *continuity property* in [CohGol77]). Since we do not have this requirement we cannot derive such a result for arbitrary X-automata. Instead, we need the following notion which previously has been quite helpful in complementing the *finitary* languages accepted by deterministic X-automata (cf. [EngVog87]).

5.5. Definition. Let $\mathrm{X} = (C, C_{in}, P, F, \mu)$ be a storage type. X *with infinite look-ahead*, denoted $\mathrm{X}_{\omega LA}$, is the storage type (C, C_{in}, P', F, μ'), where $P' = P \cup \{\inf(\mathcal{A}) | \mathcal{A} \text{ is an X-automaton}\}$, with $\mu'(x) = \mu(x)$ for each $x \in P \cup F$, and $\mu'(\inf(\mathcal{A}))(c) = true$ if and only if there exists an infinite run of $\mathcal{A}$ on Λ starting from (q_{in}, Λ, c), where q_{in} is $\mathcal{A}$'s initial state. ■

5.6. Lemma. Let $L \subseteq \Sigma^\omega$.
(1) If $L \in \text{d-XL}_{\sigma,=}$, then $\Sigma^\omega \Leftrightarrow L \in \text{d-X}_{\omega LA}\text{L}_{\sigma,=}$.
(2) If $L \in \text{d-XL}_{in,\subseteq}$ then $\Sigma^\omega \Leftrightarrow L \in \text{d-X}_{\omega LA}\text{L}_{in,\sqcap}$.
(3) If $L \in \text{d-XL}_{in,\sqcap}$ then $\Sigma^\omega \Leftrightarrow L \in \text{d-X}_{\omega LA}\text{L}_{in,\subseteq}$.

Proof. Let $\mathcal{A} = (Q, \Sigma, \delta, q_{in}, c_{in})$ be a deterministic X-automaton. For $(in, \subseteq)$ and $(in, \sqcap)$-acceptance we may assume that we have a single state set D with respect to which we accept runs. If $\mathcal{A}$ has a run on each u in Σ^ω, then $\Sigma^\omega \Leftrightarrow L_{in,\sqcap}(\mathcal{A}, \{D\}) = L_{in,\subseteq}(\mathcal{A}, \{Q \Leftrightarrow D\})$. Similarly, the complement with respect to Σ^ω of the ω-language $L_{\sigma,=}(\mathcal{A}, \mathcal{D})$ is equal to $L_{\sigma,=}(\mathcal{A}, 2^Q \Leftrightarrow \mathcal{D})$, provided that for each ω-word u in Σ^ω there is a run of $\mathcal{A}$ on u. Using infinite look-ahead $\mathcal{A}$ may be transformed in such a way that it satisfies this property. We add a special state q_{fail} to $\mathcal{A}$, with transitions $(q_{fail}, a, true, q_{fail}, \Lambda)$ for each $a \in \Sigma$, to which we will lead all 'unsuccessful runs'. There are several possibilities for the behaviour of $\mathcal{A}$ on u to be 'unsuccessful'.

(a) $\mathcal{A}$ blocks due to an undefined instruction. We can avoid that by testing the instruction as follows. For $\phi \in F^*$, consider the X-automaton $\mathcal{B}(\phi)$ consisting of two states q_0 and q_{loop} (of which q_0 is initial), and having two transitions $(q_0, \Lambda, true, q_{loop}, \phi)$ and $(q_{loop}, \Lambda, true, q_{loop}, \Lambda)$. $\mathcal{B}(\phi)$ has a run on Λ starting from (q_0, Λ, c) if and only if $\mu(\phi)(c)$ is defined. Now replace in $\mathcal{A}$ each transition (q, a, β, q', ϕ) by transitions

$(q, a, \beta \wedge \inf(\mathcal{B}(\phi)), q', \phi)$ and $(q, a, \beta \wedge \neg \inf(\mathcal{B}(\phi)), q_{fail}, \Lambda)$.

(b) $\mathcal{A}$ has a run on a finite prefix of u. Replace in $\mathcal{A}$ each transition (q, a, β, q', ϕ) by transitions $(q, a, \beta \wedge \neg \inf(\mathcal{A}(q)), q', \phi)$ and $(q, a, \beta \wedge \inf(\mathcal{A}(q)), q_{fail}, \Lambda)$, where $\mathcal{A}(q)$ is the X-automaton that equals $\mathcal{A}$, except that its initial state is q.

(c) Finally $\mathcal{A}$ may block because in some ID there are no 'useful' transitions: the present configuration satisfies none of the tests of the transitions that start in the present state (with a suitable input). This case is left to the reader. ■

Note that if $\mathrm{X}_{\omega LA}$ can be simulated by X (and this holds, e.g., for X = FS and X = PD), then the previous lemma shows that d-$\mathrm{XL}_{\sigma,=}$ is closed under complement, and that d-$\mathrm{XL}_{in,\subseteq}$ and d-$\mathrm{XL}_{in,\sqcap}$ contain the complements of each others ω-languages. In the next section we need this property for a specific storage type.

6. A Universal Storage Type

In this section we show the existence of a storage type of 'maximal power'.

6.1. Definition. The *universal storage type* U equals $(, ^*, \{\Lambda\}, P, F, \mu)$, where , is a fixed infinite set of symbols, $P = \{inK | K \subseteq \Sigma^*$, for a finite $\Sigma \subseteq , \}$, $F = \{store(x) | x \in , ^*\}$, and, for $c \in , ^*$, $\mu(inK)(c) = true$ iff $c \in K$, and $\mu(store(x))(c) = cx$. ■

6.2. Lemma. For every storage type X, U can simulate X, in the sense that $\mathrm{XL}_{\sigma,\rho} \subseteq \mathrm{UL}_{\sigma,\rho}$, and similarly for d- and r-.

Proof. Let $\mathrm{X} = (C, C_{in}, P, F, \mu)$ and let $\mathcal{A}$ be a X-automaton with initial configuration c_{in}. We will construct a U-automaton $\mathcal{A}'$ with the same behaviour. We use the configurations of U to store the sequences of instruction symbols that are performed by $\mathcal{A}$ and encode those sequences that lead to configurations in which a given test is sat-

isfied in a suitable language. Let $F_{\mathcal{A}}$ be the (finite) subset of F of instruction symbols that are used in $\mathcal{A}$. Without restriction we may assume that $F_{\mathcal{A}}$ is included in , , the alphabet of U. Let, for $\phi \in F^*$, $\mathrm{Def}(\phi) = \{\psi \in F_{\mathcal{A}}^* | \mu(\psi \cdot \phi)(c_{in}) \text{ is defined}\}$ and, for $\beta \in \mathrm{BE}(P)$, let $\mathrm{True}(\beta) = \{\psi \in F_{\mathcal{A}}^* | c = \mu(\psi)(c_{in}) \text{ is defined and } \mu(\beta)(c) = \mathit{true}\}$. Now $\mathcal{A}'$ is obtained by replacing every transition (q, a, β, q', ϕ) of $\mathcal{A}$ by the transition $(q, a, \mathit{in}\mathrm{True}(\beta) \wedge \mathit{in}\mathrm{Def}(\phi), q', \mathit{store}(\phi))$. ■

6.3. Theorem. (1) $\text{d-UL}_{ran,\subseteq} = \mathcal{F}$,
(2) $\text{d-UL}_{ran,\sqcap} = \mathcal{F} \wedge \mathcal{G}$,
(3) $\text{d-UL}_{ran,=} = \mathcal{B}(\mathcal{F})$,
(4) $\text{d-UL}_{in,\subseteq} = \mathcal{F}_\sigma$,
(5) $\text{d-UL}_{in,\sqcap} = \mathcal{G}_\delta$, and
(6) $\text{d-UL}_{in,=} = \mathcal{B}(\mathcal{F}_\sigma)$.

Proof. The inclusions from left to right follow from Corollary 5.2.
(1,5) Let $\mathcal{A}$ be the U-automaton with transitions $(q_i, a, \mathit{in}K, q_1, \mathit{store}(a))$ and $(q_i, a, \neg \mathit{in}K, q_0, \mathit{store}(a))$ for $a \in \Sigma$, $i \in \{0, 1\}$, and with initial state q_1. For this automaton $L_{in,\sqcap}(\mathcal{A}, \{\{q_1\}\}) = \mathit{lim}(K)$. This shows $\mathcal{G}_\delta \subseteq \text{d-UL}_{in,\sqcap}$. If $K = \mathit{pref}(K)$, then $L_{ran,\subseteq}(\mathcal{A}, \{\{q_1\}\}) = \mathit{adh}(K)$, which shows $\mathcal{F} \subseteq \text{d-UL}_{ran,\subseteq}$.

(2) Let $\mathcal{B}$ be the U-automaton with transitions $(q_0, a, \neg \mathit{in}L, q_0, \mathit{store}(a))$, $(q_0, a, \mathit{in}L, q_1, \mathit{store}(a))$, and $(q_1, a, \mathit{in}K, q_1, \mathit{store}(a))$ for $a \in \Sigma$, and with initial state q_0. If $K = \mathit{pref}(K)$, then $L_{ran,\sqcap}(\mathcal{A}, \{\{q_1\}\}) = L \cdot \Sigma^\omega \cap \mathit{adh}(K)$.

(3) For $\mathcal{B}(\mathcal{F}) \subseteq \text{d-UL}_{ran,=}$ it suffices to show, by (1) above, that $\text{d-UL}_{ran,=}$ is closed under complement and union. The closure of $\text{d-UL}_{ran,=}$ under complement follows from Lemma 5.6 and Lemma 6.2. The closure of $\text{d-UL}_{ran,=}$ under union can be shown using an easy direct product construction (using the fact that U can simulate U × U).

(4) $\mathcal{F}_\sigma \subseteq \text{d-UL}_{in,\subseteq}$ follows by complementation from (5) above (see Lemma 5.6).

(6) For $\mathcal{B}(\mathcal{F}_\sigma) \subseteq \text{d-UL}_{in,=}$ we use an argumentation analogous to (3) above. ■

Note that these classes are related by the diagram in Theorem 5.4.

Thus in the deterministic case the maximal power of U can be expressed as a topological family, depending on the acceptance criterion. For finitary languages U is of no interest: every finitary language can be accepted by a deterministic real-time U-automaton.

6.4. Theorem. (1) dr-UT = d-UT equals the family of continuous functions with domain in $\mathcal{G}_\delta$.
(2) dr-UT$_\omega$ = d-UT$_\omega$ equals the family of continuous functions with domain in $\mathcal{F}$.

Proof. Recall that the function $f : \Sigma^\omega \to \Delta^\omega$ is continuous in a word u if for each $m \in \mathbf{N}$ there exists $n \in \mathbf{N}$ such that $f(u[n] \cdot \Sigma^\omega) \subseteq f(u)[m] \cdot \Delta^\omega$. If $(u, v) \in T(\mathcal{M})$ for some deterministic transducer $\mathcal{M}$, and $\mathcal{M}$ outputs the first m symbols of v on the first n symbols of u, then $T(\mathcal{M})(u[n] \cdot \Sigma^\omega) \subseteq v[m] \cdot \Delta^\omega$, where Σ and Δ are the input alphabet and output alphabet of $\mathcal{M}$. Hence $T(\mathcal{M})$ is continuous. Regarding the domain of transductions it can be shown that dom(d-UT) = d-UL$_{in,\sqcap}$ = $\mathcal{G}_\delta$, and dom(d-UT$_\omega$) = d-UL$_{ran,\subseteq}$ = $\mathcal{F}$. We omit the proof of the reverse inclusions. ■

6.5. Theorem. UL$_{\sigma,\rho}$ equals the family of continuous images of $\mathcal{G}_\delta$-sets.

Proof. By Theorem 3.7 and Lemma 6.2 the UL$_{\sigma,\rho}$ are all equal. It is easy to see that UL$_{in,\sqcap} \subseteq$ dr-FST(d-UL$_{in,\sqcap}$). Hence each set in UL$_{in,\sqcap}$ is the continuous image of the intersection of two $\mathcal{G}_\delta$-sets (the domain of the transducer and the d-UL$_{in,\sqcap}$ set) which again is a $\mathcal{G}_\delta$-set. The continuous images of $\mathcal{G}_\delta$-sets are exactly the ranges of deterministic real-time U-transductions. The following easy inclusions hold: ran(dr-UT) $\subseteq$ dom(dr-UT^{-1}) $\subseteq$ dom(UT$_\omega$) $\subseteq$ UT$_\omega^{-1}$(FSL$_{in,\sqcap}$). By Theorem 3.3 this is UL$_{in,\sqcap}$. ■

Without proof we state the topological characterizations of the families r-UL$_{\sigma,\rho}$ (cf. Lemma 4.3). The same equalities hold for f-.

6.6. Theorem. r-UL$_{ran,\subseteq}$ = $\mathcal{F}$, r-UL$_{in,\subseteq}$ = $\mathcal{F}_\sigma$, and r-UL$_{in,\sqcap}$ = UL$_{in,\sqcap}$. ■

Similar results were obtained by Arnold ([Arn83], cf. [Sta84]) for the more general framework of transition systems. He discusses the acceptance types $(ran, \subseteq)$, $(in, \subseteq)$, and $(in, \sqcap)$—somewhat reformulated to deal with a possibly infinite number of states—for various kinds of transition systems. It is not difficult to see that Arnold's *deterministic*, *finitely branching*, and *countably branching* transition systems correspond in our framework closely to automata that are deterministic, have finite delay, or are unrestricted, respectively. Note that his definitions do not allow Λ-transitions, whereas in our framework the number of transitions applicable to an ID of an automaton is bounded by a constant (depending on the automaton).

Acknowledgements

We would like to thank dr. Ludwig Staiger for his useful suggestions, prof. Wolfgang Thomas for several motivating discussions, and prof. Volker Diekert for his continuous enthousiasm and for organizing this ACMICS workshop on infinite traces.

References

[Arn83] A. Arnold, *Topological characterizations of infinite behaviours of transition systems*, LNCS 154 (1983) 28-38 Springer Verlag, Berlin.

[BoaNiv80] L. Boasson & N. Nivat, *Adherences of languages*, J. Comput. System Sci. **20** (1980) 285-309.

[Büc60] J.R. Büchi, *On a decision method in restricted second order arithmetic*, in: Proc. Int. Congr. Logic, Methodology and Philosophy of Sciences 1960, Stanford University Press, Stanford, CA, 1962.

[Car84] H. Carstensen, *Fairness in deadlockfree Petri nets with the finite delay property*, Proc 5th European Workshop on appl. and theory of Petri nets, Aarhus, 1984, pp. 234-253.

[Car88] H. Carstensen, *Infinite behaviour of deterministic Petri nets*, LNCS 324 (1988) 210-219 Springer Verlag, Berlin.

[CohGol77] R.S. Cohen, A.Y. Gold, *Theory of ω-languages. II: A study of various models of ω-type generation and recognition*, J. Comput. System Sci. **15** (1977) 185-208.

[CohGol78a] R.S. Cohen, A.Y. Gold, *ω-Computations on deterministic pushdown machines*, J. Comput. System Sci. **16** (1978) 275-300.

[CohGol78b] R.S. Cohen, A.Y. Gold, *ω-Computations on Turing machines*, Theor. Comput. Sci. **6** (1978) 1-23.

[Eil74] S. Eilenberg, *Automata, languages and machines*, Ch. XIV: Infinite behavior of finite automata, Academic Press, New York and London, 1974.

[EngHoo89] J. Engelfriet, H.J. Hoogebooom, *Automata with storage on infinite words*, *in* "Proceedings ICALP 1989" (G. Ausiello, M. Dezani-Ciancaglini, S. Ronchi Della Rocca, eds.) LNCS 372 (1989) 389-303, Springer Verlag, Berlin.

[EngVog87] J. Engelfriet, H. Vogler, *Look-ahead on pushdowns*, Inform. and Computation **73** (1987) 245-279.

[Gin75] S. Ginsburg, *Algebraic and automata-theoretic properties of formal languages*, 1975, North-Holland/American Elsevier, Amsterdam/New York.

[GinGre69] S. Ginsburg, S.A. Greibach, *Abstract families of Languages*, *in* "Studies in abstract families of languages", Memoirs of the Amer.Math. Soc. 87 (1969) 1-32.

[Gre78] S.A. Greibach, *Remarks on blind and partially blind one-way multicounter machines*, Theor. Comput. Sci. **7** (1978) 311-324.

[HooRoz86] H.J. Hoogebooom, G. Rozenberg *Infinitary languages – basic theory and applications to concurrent systems*, *in* "Current trends in concurrency" (J.W. de Bakker, W.P. de Roever, G. Rozenberg, eds.) LNCS 224 (1986) 266-342, Springer Verlag, Berlin.

[Hos72] R. Hossley, *Finite tree automata and ω-automata*, MAC Technical report 102 MIT, summer 1972.

[HopUll67] J.E. Hopcroft, J.D. Ullman, *An approach to a unified theory*

of automata, The Bell System Technical Journal **XVLI** (1967) 1793-1829.

[Jan79] M. Jantzen, *On the hierarchy of Petri net Languages*, RAIRO Inf. Théor. **13** (1979) 19-30.

[Lan69] L.H. Landweber, *Decision problems for ω-automata*, Math. Systems Theor. **3** (1969) 376-384.

[LatTim86] M. Latteux, E. Timmerman, *Two characterizations of rational adherences*, Theor. Comput. Sci. **46** (1986) 101-106.

[Lin76] M. Linna, *On ω-sets associated with context-free languages*, Inform. and Control **31** (1976) 272-293.

[Mul63] D.E. Muller, *Infinite sequences and finite machines*, AIEE Proc. 4th ann. symp. switch. circ. th. and log. design, 1963, pp.3-16.

[Sco67] D. Scott, *Some definitional suggestions for automata theory*, J. Comput. System Sci. **1** (1967) 187-212.

[Sta77] L. Staiger, *Empty-storage-acceptance of ω-languages*, LNCS 56 (1977) 516-521, Springer Verlag, Berlin.

[Sta84] L. Staiger, *Projection lemmas for ω-languages*, Theor. Comput. Sci. **32** (1984) 331-337.

[Sta87] L. Staiger, *Research in the theory of ω-languages*, EIK **23** (1987) 415-439.

[StaWag74] L. Staiger, K. Wagner, *Automatentheoretische und automatenfreie Characterisierungen topologischer Klassen regulaerer Folgenmengen*, EIK **10** (1974) 379-392.

[Tho88] W. Thomas, *Automata on infinite objects*, *in* "Handbook for Theoretical Computer Science" (J. van Leeuwen, ed.), North-Holland, 1991.

[Val83] R. Valk, *Infinite behaviour of Petri nets*, Theor. Comput. Sci. **25** (1983) 311-341.

[Wag79] K. Wagner, *On ω-regular sets*, Inform. and Control **43** (1979) 123-177.

[WagSta77] K. Wagner, L. Staiger, *Recursive ω-languages*, LNCS 56 (1977) 532-537, Springer Verlag, Berlin, see also: *Rekursive Folgenmengen I*, Zeitschr. math. Logik Grundlagen Math., **24** (1978) 523-528.

On logical definability of ω-trace languages [†]

Werner Ebinger
Universität Stuttgart
Institut für Informatik
Breitwiesenstr. 20–22
D 7000 Stuttgart 80
ebinger@informatik.uni-stuttgart.de

April 1992

Abstract

Our main result is the equivalence of monadic second order logic and recognizability for languages of infinite traces. This is a generalization of the work of Thomas [Tho90b]. We propose a logical characterization that is independent of any special sort of trace automata. For another approach we use standard constructions and Büchi asynchronous cellular automata defined by Gastin and Petit [GP91].

1 Introduction

The theory of ω-trace languages is a generalization of the theory of ω-word languages [Tho90b, for an overview] and of the theory of trace languages [Maz77]. Both theories are well studied. ω-trace languages need some extra investigation, because there are new difficulties. The basic definitions of trace languages are explained well in other papers in

[†]This research has been supported by the EBRA working group No. 3166 ASMICS.

these proceedings. So we only mention some important notions in this section.

In the second section we will give a short introduction to monadic second order logic. In the third section we will prove equivalence of monadic second order logic and recognizability for ω-trace languages. This is a generalization of the corresponding results for ω-words [Tho90a, for an overview] and finite traces [Tho90b]. We will provide a new proof that does not depend on any special sort of automata for infinite traces. In the fourth section we will recall the definition of the Büchi asynchronous automata of Gastin and Petit [GP91] and give a more classical proof of the equivalence of monadic second order logic and automata for ω-trace languages. In the fifth section we want to give some remarks on first order logic on ω-traces and star-free sets. Similar ideas have been proposed independently by Hoogeboom and Thomas. However, they never were worked out extensively.

We assume that the reader is familiar with the following notions of trace languages that can be found in other papers in this proceedings. In order to extend the free partially commutative monoid $\mathbb{M}(\Sigma, D)$ [Maz77, Die90] over a dependence alphabet (Σ, D) to the infinite case, it is convenient to consider dependence graphs. In the monoid $\mathbb{G}(\Sigma, D)$ of infinite dependence graphs we consider the subset $\mathbb{R}(\Sigma, D)$ of real traces where every node has a finite past, and the submonoid $\mathbb{C}(\Sigma, D)$ which is the quotient of $\mathbb{G}(\Sigma, D)$ by the coarsest congruence which preserves real traces [Die91]. We will use $\mathbb{M}$ and $\mathbb{R}$ as abbreviations for $\mathbb{M}(\Sigma, D)$ and $\mathbb{R}(\Sigma, D)$.

A language of real traces is called *recognizable* if it is recognized by some morphism $\eta : \mathbb{M} \to S$ into a finite monoid. A real trace language $R \in \mathbb{R}$ is recognized by η if for any sequence (r_i), $r_i \in \mathbb{M}$ we have

$$r_1\, r_2\, r_3 \ldots \in R \Rightarrow \eta^{-1}(\eta(r_1))\, \eta^{-1}(\eta(r_2))\, \eta^{-1}(\eta(r_3)) \ldots \subseteq R$$

The connection between recognizability for infinite traces and infinite words is given by [Gas91]

$$R \in \mathrm{Rec}(\mathbb{R}) \iff \varphi^{-1}(R) \in \mathrm{Rec}(\Sigma^\infty),$$

where $\varphi : \Sigma^\infty \to \mathbb{R}(\Sigma, D)$ is the extension of the canonical mapping to the infinite case.

2 Monadic second order logic over ω-traces

We can represent a real trace $t \in \mathbb{R}(\Sigma, D)$ as a labeled partial order $(V, <, \ell)$. Logical *formulae* can be defined over a *structure* with *signature* $(V, <, (P_a)_{a \in \Sigma})$. We use first order variables x, y, z, ... ranging over V and monadic second order variables X, Y, Z, ... ranging over $\mathcal{P}(V)$. Formulae are defined inductively:

- *Predicates:* First order predicates of the form $x < y$, $P_a(x)$ and monadic second order predicates of the form $X(x)$ are formulae for arbitrary variables x, y, X and all $a \in \Sigma$.
- *Logical operations:* If φ and ψ are formulae, then $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, and $(\neg\varphi)$ are formulae, too.
- *Quantifiers:* If φ is a formula, then $\exists x\varphi$, $\forall x\varphi$, $\exists X\varphi$, and $\forall X\varphi$ are formulae, too.

This monadic second order logic is called MSOL. We will also write $z \in X$ instead of $X(z)$ and freely use abbreviations like $X \subseteq Y$. Of course, we can also express $x = y$ and $X = Y$ in MSOL.

Example 1 We can express the property "t begins with b" by the formula

$$\exists x(P_b(x) \wedge \forall y(x = y \vee x < y)).$$

□

In monadic second order the power of the logic does not depend on the difference between the $<$-relation and the edge relation of the Hasse diagram of $(V, <)$, which constitutes some kind of successor relation. The edge relation E of the Hasse diagram of $(V, <)$ is expressible by $<$ (even in first order logic):

$$x \; E \; y \quad \text{iff} \quad x < y \wedge \neg\exists z(x < z \wedge z < y),$$

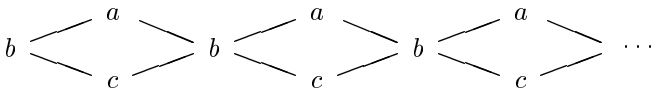
and vice versa (in monadic second order logic):

$$x < y \quad \text{iff} \quad \neg x = y \wedge$$

$$\forall X(x \in X \wedge \underbrace{\forall z \forall z'(z \in X \wedge z \; E \; z' \rightarrow z' \in X)}_{\substack{\text{If a node is in } X\text{, then} \\ \text{all } <\text{-greater nodes are in } X\text{, too.}}} \rightarrow y \in X)$$

Now we are free to use both, the $<$-relation and the edge relation E in monadic second order formulae.

Example 2



Traces of the language $T = (bac)^\omega$ over the dependence alphabet D : a — b — c are described by the following formula. The parts in quotes can be easily replaced by a proper formula (even in first order).

"There is exactly one minimal node, labeled b."
$\wedge$ "Every b-node has two successors, labeled a and c"
$\wedge$ "Every b-node except the minimal one has two predecessors, labeled a and c"
$\wedge$ "Every a-node has exactly one predecessor and one successor, both labeled b"
$\wedge$ "Every c-node has exactly one predecessor and one successor, both labeled b"

□

In the fifth section we will go from formula to automata. These constructions can be simplified if we first reduce the logic MSOL to the *restricted monadic second order logic* MSOL_0. The logic MSOL_0 is the logic MSOL with the restriction, that there are no first order variables. Instead of first order predicates we use only second order predicates of the form $X_i \subseteq X_j$, $X_i < X_j$, and $X_i \subseteq P_a$. Here $X_i < X_j$ means X and Y are singleton sets and the element of X is smaller than the element of

Y. As formulae grow really huge using only these basic predicates, we give the following abbreviations:

$$
\begin{array}{lll}
X = Y & \text{for} & X \subseteq Y \wedge Y \subseteq X \\
X \neq Y & \text{for} & \neg X = Y \\
Sing(X) & \text{for} & \exists Y (\overbrace{Y \subseteq X \wedge Y \neq X}^{\text{“proper subset”}} \wedge \\
 & & \underbrace{\neg \exists Z (Z \subseteq X \wedge Z \neq X \wedge Z \neq Y)}_{\text{“exactly one”}})
\end{array}
$$

We now describe how to eliminate first order variables by giving an example. A formula like

$$\underline{\forall x}\, \exists Y (x \in Y \wedge \underline{\exists z}(x < z \wedge P_a(z))$$

is expressed as

$$\underline{\forall X (Sing(X) \rightarrow}\, \exists Y (X \subseteq Y \wedge \underline{\exists Z (Sing(Z) \wedge}\, X < Z \wedge Z \subseteq P_a)).$$

3 Equivalence of recognizability and logic

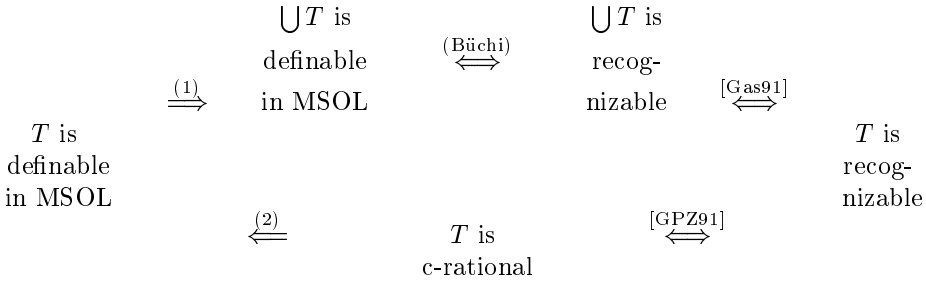
In this section we prove the equivalence of recognizability and logic. Our proof is independent of any special automata model for (infinite) traces. In the proof we will use *c-rational trace languages* [Och85, GPZ91] (also called co-rational). c-rational languages are defined inductively:

- All *finite trace languages* $A \subseteq \mathbb{M}$ of finite traces are c-rational.
- If $A \subseteq \mathbb{M}$, $B \subseteq \mathbb{M}$, and $C \subseteq \mathbb{R}$ are c-rational, then the *concatenation* $A\,B$ and $A\,C$ is c-rational.
- If $A \subseteq \mathbb{M}$ is c-rational, then the concurrent iteration A^{c*} (called coiteration by Ochmansky) and the concurrent ω-iteration $A^{c\omega}$ are c-rational. For the concurrent iteration independent (connected) parts are iterated concurrently. This means A^{c*} ($A^{c\omega}$) in the usual star (omega) iteration over $C(A) = \{\ t \in \mathbb{M} \mid t$ is a connected component of some $s \in A\ \}$.

We can also use a constraint version of c-iteration were only connected parts may be iterated. This gives the same class of c-rational languages.

Theorem 3 *A trace language $T \subseteq \mathbb{R}(\Sigma, D)$ is recognizable if and only if T is definable in monadic second order logic.*

Proof: The following figure shows what we have to prove. $\bigcup T$ denotes the union of all traces $t \in T$, that is the word language that contains all members of traces in T. For the canonical mapping $\varphi : \Sigma^\infty \to \mathbb{R}$ we have $\bigcup T = \varphi^{-1}(T)$.



(1): We can not use the same formula for both, traces and words, as the underlying interpretation of the order is different. For example the formula

$$\exists x, y, z\ P_a(x) \wedge P_c(y) \wedge P_b(z) \wedge x < z \wedge y < z \wedge (x < y \vee y < x)$$

over the dependence alphabet D : $a - b - c$ is not true for the trace acb, but it is true for both words in this trace. But it suffices to express the partial order on the traces by the linear order on the words. This can be done by one single first order formula, but in order to render it easier to follow, we proceed in two steps. We can first express the partial order by the edge relation E of the Hasse diagram of the dependence graph. This was described in the section on monadic second order logic. Then we express the edge relation E by the linear order on the words: $x\ E\ y$ if and only if

$$\bigvee_{(a,b) \in D} (Q_a(x) \wedge Q_b(y) \wedge x < y$$

$$\wedge\, \forall z\, (x < z \wedge z < y \rightarrow \bigwedge_{\substack{c \text{ with} \\ (a,c) \in D \text{ or} \\ (b,c) \in D}} \neg Q_c(z))).$$

Now if a trace t satisfies a formula φ, then all words w in the trace satisfy the new formula φ', which we obtained by replacing the predicates $x\ E\ y$ by the formula above. And vice versa if a word $w \in t$ satisfies the new formula φ', then all words $w' \in t$ in the same trace satisfy this new formula φ', and the trace t containing this word w satisfies the original formula φ. We can formalize this using $\models$ as abbreviation of "is a model for"

$$t \models \varphi \iff \exists w \in t\ w \models \varphi' \iff \forall w \in t\ w \models \varphi'.$$

(2): For this implication of the diagram we perform an induction over the construction of concurrent rational (c-rational) expressions.

- A is a finite set of traces: We give a formula φ_t for every single word in A and combine them in a disjunction
$$\varphi_A = \bigvee_{t \in A} \varphi_t.$$
- $A \cup B$ for c-rational sets A and B: Combine the formulae φ_A and φ_B for A and B to
$$\varphi_{A \cup B} = \varphi_A \vee \varphi_B.$$
- $A \cdot B$ for c-rational sets A and B: For the formulae φ_A and φ_B we define formulae with restricted quantification. $\varphi_A|_X$ for some set X is the formula φ_A where we replace every subformula $\exists x \psi$ by $\exists x\ (x \in X \wedge \psi)$, $\forall x \psi$ by $\forall x\ (x \in X \rightarrow \psi)$, $\exists Y \psi$ by $\exists Y\ (Y \subseteq X \wedge \psi)$, and $\forall Y \psi$ by $\exists Y\ (Y \subseteq X \rightarrow \psi)$. The formula $\varphi_{A \cdot B}$ is defined as

  possible max./min. elements of A/B:
  $$\bigvee_{\substack{\{a_1,\ldots,a_k\} \\ \subseteq \Sigma}} \bigvee_{\substack{\{b_1,\ldots,b_l\} \\ \subseteq \Sigma}} \exists x_{a_1} \ldots \exists x_{a_k} \exists x_{b_1} \ldots \exists x_{b_l}$$

  incomparable:
  $$\Big(\bigwedge_{\substack{1 \le i \le k \\ 1 \le j \le k \\ i \neq j}} (\neg\, x_{a_i} < x_{a_j} \wedge \neg\, x_{b_i} < x_{b_j}) \wedge$$

nothing between A and B

$$\bigwedge_{\substack{1 \le i \le k \\ 1 \le j \le l}} \neg\exists z\,(x_{a_i} < z \wedge z < x_{b_j}) \wedge$$

every node belongs to a trace

$$\forall z \bigvee_{\substack{1 \le i \le k \\ 1 \le j \le l}} \Big(z < x_{a_i} \vee z = x_{a_i} \vee x_{b_j} < z \vee x_{b_j} = z \Big) \wedge$$

$$\varphi_A\big|_{\{x \mid \bigvee_{1 \le i \le k} x \le x_{a_i}\}} \wedge \varphi_B\big|_{\{x \mid \bigvee_{1 \le j \le l} x_{b_j} \le x\}} \Big)$$

- A^{c*}, $A^{c\omega}$ for a c-rational set A, where only connected parts are iterated: The logical definability of A^{c*} was already claimed by Thomas [Tho90b]. We define $\varphi_{A^{c*}}$ and $\varphi_{A^{c\omega}}$ as

$$\exists X_{min} \exists X_{max}$$

For every node x find borders of subtrace

$$\forall x \Bigg(\bigvee_{\substack{\{a_1,\ldots,a_k\} \\ \subseteq \Sigma}} \bigvee_{\substack{\{b_1,\ldots,b_l\} \\ \subseteq \Sigma}} \exists x_{a_1} \ldots \exists x_{a_k} \exists x_{b_1} \ldots \exists x_{b_l}$$

$$\Big(x_{a_1} \in X_{min} \wedge \ldots \wedge x_{a_k} \in X_{min} \wedge$$

starting/ending nodes of the subtrace are incomparable and maximal such sets

$$x_{b_1} \in X_{max} \wedge \ldots \wedge x_{b_l} \in X_{max} \wedge$$

"$x_{a_1} \ldots x_{a_k}$ are a maximal antichain in X_{min} ($x_{a_1} \ldots x_{a_k}$ are incomparable and there's no node in X_{min}, that is incomparable to $x_{a_1} \ldots x_{a_k}$)" $\wedge$

"$x_{b_1} \ldots x_{b_l}$ are a maximal antichain in X_{max}" $\wedge$

subtrace around x well bounded

$$\bigwedge_{1 \le i \le k} x_{a_i} \le x \wedge \bigwedge_{1 \le j \le l} x \le x_{b_j} \wedge \neg\exists z$$

$$\Big(\big((z < x \wedge \bigvee_{1 \le i \le k} x_{a_i} < z) \vee (x < z \wedge \bigvee_{1 \le j \le l} z < x_{b_j}) \big) \wedge$$

$$(z \in X_{min} \vee z \in X_{max}) \Big) \wedge$$

$$\text{Apply } \varphi_A \text{ to subtrace} \qquad \varphi_A|_{\geq\{x_{a_i}\},\leq\{x_{b_j}\}}\Big)\Big) \wedge$$

$$\forall x(\text{“}x \text{ minimal in the whole trace”}) \rightarrow x \in X_{min}) \wedge$$

$$\text{for } A^{c*}: \qquad \forall x(\text{“}x \text{ maximal in the whole trace”}) \rightarrow x \in X_{max}) \wedge$$

$$\bigvee_{1\leq i\leq|\Sigma|} (\exists x_1 \ldots \exists x_i \; \forall x \bigvee_{1\leq j\leq i} x \leq x_j)$$

$$\text{for } A^{c\omega}: \qquad \forall x \exists y \; x < y$$

□

4 Büchi Asynchronous Cellular Automata and equivalence to logic

Büchi asynchronous automata and Büchi asynchronous cellular automata were defined by Gastin and Petit [GP91]. These are generalizations of Büchi automata on infinite words and asynchronous (cellular) automata defined by Zielonka [Zie87]. The recognition power of Büchi asynchronous automata is equal to the recognition power of Büchi asynchronous cellular automata, like it is equal in the finite case for asynchronous automata and asynchronous cellular automata. In this paper we restrict ourselves to Büchi asynchronous cellular automata. The proofs are almost the same for Büchi asynchronous automata, and some details might be even easier.

Definition 4 *A* Büchi asynchronous cellular automaton $\mathcal{A}$ *is a tuple* $\mathcal{A} = ((Q_a)_{a\in\Sigma}, (\delta_a)_{a\in\Sigma}, S, F, \mathcal{R})$ *where* Q_a *is the local state set for process* a, $\delta_a : \prod_{i\in D(A)} Q_i \rightarrow \mathcal{P}(Q_a)$ *is the local transition function for process* a, $S \subseteq \prod_{a\in\Sigma} Q_a$ *is the set of starting states,* $F \subseteq \prod_{a\in\Sigma} Q_a$ *is the set of accepting states for finite traces and* $\mathcal{R} \subseteq \prod_{a\in\Sigma} \mathcal{P}(Q_a)$ *is the acceptance table for infinite traces.*

In order to define runs of such an automaton we consider a trace $t \in \mathbb{R}(\Sigma, D)$ as a *labeled partial order* $(V, <, \ell)$ with $\ell : V \rightarrow A$. A *run*

r of $\mathcal{A}$ on t is a function $r : V \cup \{v_0\} \to (\bigcup_{a\in\Sigma} Q_a) \cup S$, where v_0 is an additional $<$-minimal node v_0, that is labeled with a starting state $r(v_0) \in S$, and for all $v \in V$:

$$r(v) \in \delta_{\ell(v)}((r(v^i_{max}))_{i\in D(\ell(v))}),$$

where v^i_{max} is the $<$-maximal node labeled i before v. The set of *locally repeated states* is defined as

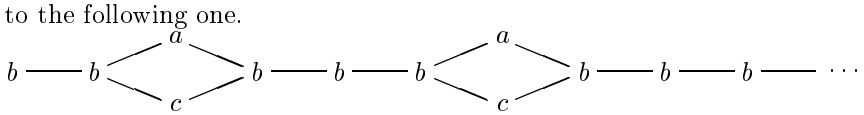
$$\begin{aligned} f_a(r) &:= \{\, q \in Q_a \mid \exists^\omega v : (\ell(v) = a \wedge r(v) = q) \\ &\qquad \vee \exists v(\neg\exists w(v < w \wedge \ell(w) = a) \wedge r(v) = q) \,\}. \qquad (1) \\ f(r) &:= \prod_{a\in\Sigma} f_a(r) \qquad (2) \end{aligned}$$

is the tuple of *repeated states*. The run r *accepts* t if $f(r(t)) \supseteq R$ for some $R \in \mathcal{R}$ or if the maximal nodes of a finite run form a state in F. The accepted trace language of $\mathcal{A}$ is

$$T(\mathcal{A}) = \{\, t \mid \text{there exists an accepting run } r \,\}.$$

Gastin and Petit [GP91] proved, that Büchi asynchronous cellular automata accept exactly the recognizable ω-trace languages. These languages are closed under intersection $\cap$, union $\cup$, complement $\bar{}$ and the concurrent star operation ^{c*}.

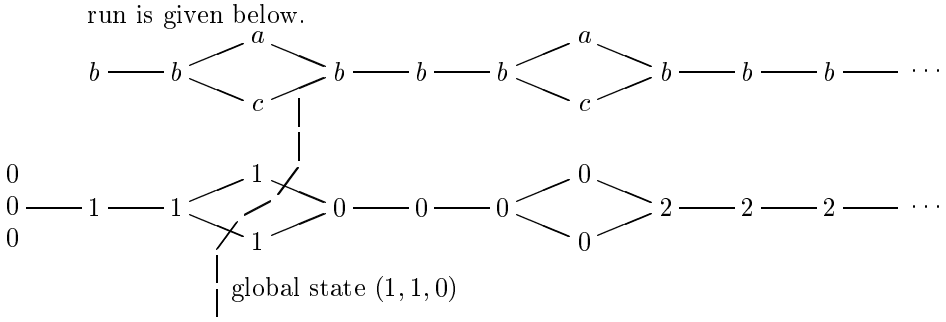
Example 5 The dependence graphs of traces in the language $T = (b^*bac)^*b^\omega$ with the dependence alphabet D : a — b — c look similar to the following one.



An automaton that accepts T is $\mathcal{A} = ((Q_a)_{a\in\Sigma}, (\delta_a)_{a\in\Sigma}, S, F, \mathcal{R})$ with $Q_a = \{0,1\}$, $Q_b = \{0,1,2\}$, $Q_c = \{0,1\}$, and the transition function

$$\begin{array}{lllll} \delta_a: & (0,1) & \mapsto & \{1\} & \\ & (1,0) & \mapsto & \{0\} & \\ \delta_c: & (1,0) & \mapsto & \{1\} & \\ & (0,1) & \mapsto & \{0\} & \end{array} \qquad \begin{array}{lllll} \delta_b: & (0,0,0) & \mapsto & \{1,2\} \\ & (0,1,0) & \mapsto & \{1\} \\ & (1,1,1) & \mapsto & \{0,2\} \\ & (1,0,1) & \mapsto & \{0\} \\ & (0,2,0) & \mapsto & \{2\} \\ & (1,2,1) & \mapsto & \{2\} \end{array}$$

There are two possibilities to define the acceptance table: $\mathcal{R} = \{(\{0\},\{2\},\{0\}),(\{1\},\{2\},\{1\})\}$ or $\mathcal{R}' = \{(\emptyset,\{2\},\emptyset)\}$. An example for a run is given below.



□

The equivalence of automata and logic is a consequence of the equivalence of recognizability and logic, since Büchi asynchronous cellular automata accept exactly the recognizable languages. In this section we will use the classical approach and provide an independent proof. We use standard techniques and some special constructions for Büchi asynchronous cellular automata. Since the case of finite traces has already been solved [Tho90b] we restrict ourselves to the case of infinite traces.

Theorem 6 *A trace language $T \subseteq \mathbf{R}(\Sigma, D)$ is accepted by some Büchi asynchronous cellular automaton if and only if T is definable in monadic second order logic.*

Proof: "⇒": Let $\mathcal{A} = ((Q_a)_{a\in\Sigma}, (\delta_a)_{a\in\Sigma}, S, F = \emptyset, \mathcal{R})$ with $Q_a = \{0,\ldots,m_a\}$ be a Büchi asynchronous cellular automaton which accepts T. Every trace t is considered as a structure $(V, <, \ell)$. A computation r corresponds to a starting state $s \in S$ and sets $Y_{a0}, Y_{a1}, \ldots, Y_{am_a}$ (for every $a \in \Sigma$), where $Y_{ai} = \{\ v \in V \mid \ell(v) = a \wedge r(v) = i\ \}$ contains all nodes that are labeled a and have the state i in the run r. The formula is given as follows.

existence of a computation $\bigvee_{s\in S} \exists Y_{a_1 0} \ldots \exists Y_{a_1 m_{a_1}}$

$$
\begin{array}{ll}
 & \vdots \\
 & \exists Y_{a_n 0} \ldots \exists Y_{a_n m_{a_n}} \\
\text{every node gets at most one state} & \Big(\text{“disjunct sets”} \wedge \\
\text{labeling according to transition table} & \forall v \bigvee_{\substack{(p,a,q) \\ q \in \delta_a(p)}} \Big(P_a(v) \wedge v \in Y_{aq} \wedge \\
\text{find preceding state} & \bigwedge_{b \in D(a)} \Big(\exists w \, \big(P_b(w) \wedge w < v \wedge \\
 & \quad \neg \exists u \, (w < u \wedge u < v \wedge P_b(u)) \wedge \\
 & \quad Y_{b\pi_b(p)}(w) \big) \vee \\
\text{or the starting state} & \quad \big(\neg \exists w \, (P_a(w) \wedge w < v) \wedge \\
 & \quad \text{“}\pi_b(s) = \pi_b(p)\text{”} \big) \Big) \Big) \wedge \\
\text{choose an element of the acceptance table} & \bigvee_{R = \prod_{a \in \Sigma} R_a \in \mathcal{R}} \quad \bigwedge_{a \in \Sigma} \quad \bigwedge_{\substack{i=1 \\ R_a = \{q_1, \ldots, q_l\}}}^{l} \\
\text{infinite repetition} & \quad \Big(\exists X (\text{“}X \text{ is infinite.”} \wedge X \subseteq Y_{aq_i}) \vee \\
\text{only finitely many } a & \quad \exists v \; v \in Y_{aq_i} \wedge \neg \exists w \, (v < w \wedge P_a(w)) \Big) \Big)
\end{array}
$$

“$\Leftarrow$”: Let φ be a second order formula that is in the restricted form of MSOL_0 . Without loss of generality we have to consider predicates $X \subseteq Y$, $X < Y$, and $X \subseteq P_a$. We are going to perform induction over the construction of formulae. As there appear formulae with free variables in this induction we have to consider formulae $\varphi(X_1, \ldots, X_n)$ with free variables $X_1, \ldots, X_n$ over the dependence alphabet $(\Sigma \times \{0,1\}^n, D_n)$ with

$$(a, \cdot) \; D_n \; (b, \cdot) \quad \text{iff} \quad a \; D \; b.$$

Induction foundation: We are working with formulae $\varphi(X, Y)$ with two free variables X, Y. In this case we use an alphabet $\Sigma \times \{0,1\}^2$. The first additional digit indicates membership in X and the second additional digit indicates membership in Y. The generalization to arbitrary many free variables is easy. For all $x \in \Sigma$ we consider processes $x00$, $x01$, $x10$, and $x11$. That gives a number of $4|\Sigma|$ processes.

- $X \subseteq Y$: This is a very simple automaton with only one global state $0^{4|\Sigma|} \in S$. The processes $x00$, $x01$, and $x11$ continue and stay in the local state 0, if they read a node labeled $x00$, $x01$, and $x11$. These are nodes that are in Y if they are in X. The processes $x10$ stop, because they read nodes that are in X, but not in Y. The acceptance table is $\mathcal{R} = (\{0\}, \ldots, \{0\})$.

$$\begin{aligned} \delta_{x00}(0^{D(x00)}) &= \{0\} \\ \delta_{x01}(0^{D(x00)}) &= \{0\} \\ \delta_{x11}(0^{D(x00)}) &= \{0\} \\ \delta_{x10}(0^{D(x00)}) &= \emptyset \qquad \text{for all } x \in \Sigma \end{aligned}$$

- $X < Y$: Again we start out with the global starting state $0^{4|\Sigma|}$. The local state 0 indicates a process not having any information. 1 indicates a process suggesting that it has read the first node in X, 2 means that there was a node in X somewhere, and 3 stands for reading a node in Y after there has been a node somewhere in X. Every process $x00$ remains in state 0, if all dependent states are still in state 0. Every process $x10$ waits until it is active and then changes its local state to 1, if all dependent processes are still in state 0. Now the automaton has read a node that is in X but not in Y. Every process $x00$ that sees a 1 or 2 in a dependent state and is still in state 0 changes its own state to 2. In doing this they spread the news that there has been a node in X. A process $x01$ that sees a 1 or 2 in a dependent state changes its own state to 3, to indicate that it has read a node in Y that is $<$-greater than the first node that was in X. $\mathcal{R} = \{ \prod_{a \in \Sigma} \mid |R_a| = 1 \text{ for all } a \in \Sigma \text{ and exactly once } R_a = \{1\} \text{ and exactly once } R_a = \{3\} \}$.

$$\begin{aligned}
\delta_{x00}(0^{D(x00)}) &= \{0\} \\
\delta_{x00}(\underbrace{\{0,1,2\}^{D(x00)}}_{\text{some 1 or 2, } q_{x00}=0}) &= \{2\} \\
\delta_{x10}(0^{D(x00)}) &= \{1\} \\
\delta_{x01}(\underbrace{\{0,1,2\}^{D(x00)}}_{\text{some 1 or 2, } q_{x01}=0}) &= \{3\} \\
\delta_{x00}(\underbrace{\{0,1,2,3\}^{D(x00)}}_{q_{x00}=q}) &= \{q\} \quad q \in \{1,2,3\}
\end{aligned}$$

- $X \subseteq P_a$: Now we have only one free variable and we start from $0^{2|\Sigma|} \in S$. All processes $x0$ (for all $x \in \Sigma$) and the process $a1$ continue and stay in the local state 0. All other processes stop. The acceptance table is $\mathcal{R} = \{\{0\}, \ldots, \{0\}\}$.

$$\begin{aligned}
\delta_{x0}(0,\ldots,0) &= \{0\} \quad \text{for all } x \in \Sigma \\
\delta_{a1}(0,\ldots,0) &= \{0\}
\end{aligned}$$

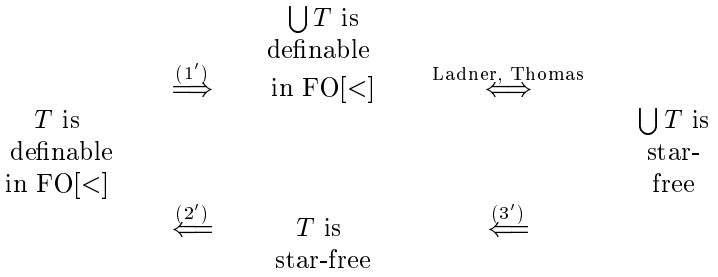
Induction step:

- Closure under conjunction $\wedge$ and disjunction $\vee$: Cartesian product of automata [GP91].
- Closure under complement $\neg$: Büchi asynchronous automata recognize exactly the recognizable languages [GP91] and these are closed under complement.
- Closure under quantification (projection) of variables $\exists$ ($\forall$ may be expressed by $\neg$ and $\exists$): The automaton for a formula $\exists X \varphi(X)$ (we can easily add more free variables, that are not quantified in this step) over the alphabet $\Sigma \times \{0,1\}$ is built taking the automaton for $\varphi(X)$. We melt every pair of processes $x0, x1$. The new process x has state set $Q_x = Q_{x0} \times Q_{x1}$ and guesses nondeterministically whether a node is in X or not and then performs a transition either in the component Q_{x0} or in the component Q_{x1}. □

If we had a construction of a complement automaton, our proof would be completely constructive. A complementation construction similar to the standard complementation construction for Büchi automata does work. But we are looking for a more constructive complementation.

5 Star-free ω-trace languages and first order logic

In the case of finite traces [GRS91], finite words, and ω-words [Tho90a, for an overview] star-free languages are exactly the first order definable ones. In order to investigate star-free ω-trace languages we first have to give a definition of star-free ω-trace languages. One possibility is to use star-free expressions with only boolean operations and concatenation, where the complement can be taken with respect to A^* in order to get star-free sets of finite traces and with respect to A^* and A^ω in order to get star-free sets of infinite traces. We can perform a part of the proof of the equivalence of recognizability and logic in the case of star-free languages and first order logic:



For the implication (1′) we have to express the $<$-relation on the traces directly by a first order formula using the $<$-relation on the words: $x < y$ in the partial order on the traces if and only if

$$\bigvee_{\substack{(a_1,a_2)\in D\\(a_2,a_3)\in D\\ \vdots \\(a_{l-1},a_l)\in D\\ \{a_1,\ldots,a_l\}\subseteq A\\ |\{a_2,\ldots,a_l\}|=l-1}} \exists x_2 \ldots \exists x_{l-1} \big(Q_{a_1}(x) \wedge Q_{a_2}(x_2) \wedge \ldots \wedge Q_{a_{l-1}}(x_{l-1}) \wedge Q_{a_l}(y) \wedge x < x_2 < \ldots < x_{l-1} < y \big)$$

For the implication (2′) we replace the boolean operations by the corresponding logical operations. For concatenation we can use the first order formula that we gave in the proof of equivalence of logic and recognizability. The remaining direction (3′) will be shown in a forthcoming joint work with V. Diekert and A. Muscholl.

Acknowledgements

The author wants to thank Volker Diekert, Paul Gastin, Hendrik Jan Hoogeboom, Anca Muscholl, Antoine Petit, and Wolfgang Thomas for many helpful discussions, corrections, and suggestions.

References

[Die90] V. Diekert. *Combinatorics on Traces.* Number 454 in Lecture Notes in Computer Science. Springer, Berlin-Heidelberg-New York, 1990.

[Die91] V. Diekert. On the concatenation of infinite traces. In Choffrut C. et al., editors, *Proceedings of the 8th Annual Symposium on Theoretical Aspects of Computer Science (STACS'91), Hamburg 1991*, number 480 in Lecture Notes in Computer Science, pages 105–117. Springer, Berlin-Heidelberg-New York, 1991.

[Gas91] P. Gastin. Recognizable and rational trace languages of finite and infinite traces. In Choffrut C. et al., editors, *Proceedings of the 8th Annual Symposium on Theoretical Aspects of Computer Science (STACS'91), Hamburg 1991*, number 480 in Lecture Notes in Computer Science, pages 89–104. Springer, Berlin-Heidelberg-New York, 1991.

[GP91] Paul Gastin and Antoine Petit. Asynchronous automata for infinite traces. Rapport de Recherche 707, Université de Paris-Sud, 1991. To appear in *ICALP 1992*, also this volume, pages 29–45.

[GPZ91] P. Gastin, A. Petit, and W. Zielonka. A Kleene theorem for infinite trace languages. In J. Leach Albert et al., editors, *Proceedings of the 18th International Colloquium on Automata Languages and Programming (ICALP'91), Madrid (Spain) 1991*, number 510 in Lecture Notes in Computer Science, pages 254–266. Springer, Berlin-Heidelberg-New York, 1991.

[GRS91] Giovanna Guaiana, Antonio Restivo, and Sergio Salemi. On aperiodic trace languages. In Choffrut C. et al., editors, *Proceedings of the 8th Annual Symposium on Theoretical Aspects of Computer Science (STACS'91), Hamburg 1991*, number 480 in Lecture Notes in Computer Science, pages 76–88. Springer, Berlin-Heidelberg-New York, 1991.

[Maz77] A. Mazurkiewicz. Concurrent program schemes and their interpretations. DAIMI Rep. PB 78, Aarhus University, Aarhus, 1977.

[Och85] E. Ochmanski. Regular behaviour of concurrent systems. *Bulletin of the European Association for Theoretical Computer Science (EATCS)*, 27:56–67, Oct 1985.

[Tho90a] Wolfgang Thomas. Automata on infinite objects. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 4, pages 133–191. Elsevier Science Publishers B. V., 1990.

[Tho90b] Wolfgang Thomas. On logical definability of trace languages. In V. Diekert, editor, *Proceedings of a workshop of the ESPRIT Basic Research Action No 3166: Algebraic and Syntactic Methods in Computer Science (ASMICS), Kochel am See, Bavaria, FRG (1989)*, Report TUM-I9002, Technical University of Munich, pages 172–182, 1990.

[Zie87] W. Zielonka. Notes on finite asynchronous automata. *R.A.I.R.O.-Informatique Théorique et Applications*, 21:99–135, 1987.

On Temporal Logics for Trace Systems

Wojciech Penczek*
Institute of Computer Science, Polish Academy of Sciences
00-901 Warsaw, PKiN, P.O. Box 22, Poland

March 1992

Abstract

We investigate an extension of CTL (Computation Tree Logic) by past modalities, called CTL_P, interpreted over Mazurkiewicz's trace systems. The logic is powerfull enough to express most of the partial order properties of distributed systems like serializability of database transactions, snapshots, parallel execution of program segments, or inevitability under concurrency fairness assumption. We show that a model checking problem for the logic is NP-hard, even if past modalities cannot be nested. Then, we give a one exponential time model checking algorithm for the logic without nested past modalities. We show that all the interesting partial order properties can be model checked using our algorithm. Then, we show that it is possible to extend the model checking algorithm to cover the whole language and its extension to CTL^*_P. At the end, we prove that the logic is undecidable and we discuss consequences of our results on using propositional versions of partial order temporal logics to synthesis of concurrent systems from their specifications.

Keywords: Concurrency; Trace Systems; Partial Order Temporal Logics; Automated Verification.

*This research has been partly supported by the Netherlands grant NWO NF 3/62 - 500 and a grant from The Wolfson Research Awards Scheme in The United Kingdom.

1 Introduction

Linear time [1] and branching time [2] temporal logics are usually applied for specifying and proving properties of concurrent systems and programs. Lately, several attempts have been made to use also logics interpreted over partial order structures [3 - 14]. The main motivation for defining these logics was to express properties inherent in the partial order interpretations, more specifically to distinguish concurrency from non-determinism.

There are two approaches to extend linear and branching time temporal logics to partial order logics on global states. Either run modalities are introduced (see QISTL [5], ISTL [5,6], CCTL [9,15], Petri Net Logics [11,12]), or past operators over partial order semantics are defined (see POL [14], PN-logics [11,12], and [13, 16]). Temporal logics over partial order semantics allow for expressing properties not expressible in logics over interleaving semantics. These properties are: inevitability under concurrency fairness assumption [17,18], serializability of database transactions [13, 16], causal successor [11, 12, 14], or the parallel execution of program segments [13, 14].

Model checking is one of the main methods of automated verification of concurrent systems [38]. It has been intensively studied for linear-time temporal logics [19, 20], branching-time temporal logics [21 - 26], and modal μ-calculi [25,27]. Model checking has been also applied to prove properties of systems, represented by partial orders of local states [3, 4]. Methods for making it applicable to very large systems and of avoiding the state explosion problem have been proposed in [28 - 31].

As far no one has tried to investigate whether and how logics with partial order past operators, interpreted over global state models, can be used for automated verification of concurrent systems. Since Hennessy and Stirling [32] introduced backward modalities to program logics, all the work has concentrated around the induced equivalences [33,34], expressiveness issues [11,12,14,32,33], and proof systems [13,16]. In this paper we fill this gap. Moreover, our results explain why it is so difficult to deal with partial order logics, what is a common observation.

We start with defining a logic CTL_P, which is a simple extension of

CTL by past modalities. In fact, the language of CTL_P is a restriction of that, considered in [3,4,13,16], and it is an extension of the language of Hennessy-Milner logic with past modalities [32]. Our choice is motivated by the following observations. CTL has proved to be a very usefull logic for automated verification. Model checking for CTL is linear in the size of a model and linear in the lenght of the formula [21], and the complexity of checking satisfiability is deterministic exponential in the size of a tested formula [35], whereas the complexity of model checking for CTL* is PSPACE-complete [22], and checking satisfiability is deterministic double exponential [36]. It turned out that it is possible to extend CTL to fair CTL (FCTL) [24] or to model check CTL formulas over fair paths [26] without changing the complexity of CTL model checking. Moreover, different methods dealing with the state explosion problem have been given [28,29].

We would be quite happy with these results, if the logic could distinguish concurrency from non-determinism and consequently, properties of partial order executions could be expressed. Unfortunately, this is not the case. Therefore, it seems very natural to consider a minimal extension of CTL s.t. properties of partial orders can be specified and proved. This can be done by introducing past modalities to the language. Then, our logic has to be interpreted over partial order models rather than over trees. Again, we select the simplest and the most frequently used partial order structures of global states, namely, Mazurkiewicz's trace systems (see [13,17,18,37,46]). In this paper we investigate consequences of our extension. Firstly, we show that for proving all the interesting partial order properties for finite state systems we can restrict ourselves to a model checking algorithm for the language without nested past modalities, call it CTL_{P-}. Then, we prove that model checking for CTL_{P-} is NP-hard. Consequently, we give a one exponential time model checking algorithm for this restricted language and show how it can be extended (if ever needed) to cover the whole logic. Our model checking algorithm requires a new technique, not applied before to model checking for CTL or CTL*. Secondly, we turn to the problem of determining satisfiability for CTL_P formulas. To our surprise, we show that even CTL_{P-} is not decidable. Since this result can be extended as well on other par-

tial order temporal logics, interpreted over trace systems, like ISTL [13], or the logic, defined in [16], therefore we show an important limitation in applying partial order logics to synthesis of concurrent systems from their specifications.

The rest of this paper is organized as follows. In section 2 trace systems are introduced. EN - systems and trace semantics is defined in section 3. Then, in section 4 the logic CTL_P and its semantics is presented. Acceptors for finite state trace systems and model generators are defined in section 5. Section 6 contains the proof of NP-hardness of CTL_P model checking and section 7 shows a model checking algorithm for CTL_{P-} and its possible extensions. Undecidability of CTL_P is proved in section 8 and then, in section 9, CTL_P is compared with other temporal logics. Final remarks are given in section 10.

2 Trace Systems

We start with introducing notions of traces and trace systems from [17].

By an *independence alphabet* we mean any ordered pair (Σ, I), where Σ is a finite set of symbols (*action names*) and $I \subseteq \Sigma \times \Sigma$ is a symmetric and irreflexive binary relation in Σ (the *independence* relation). Let (Σ, I) be an independence alphabet. Define $\equiv$ as the least congruence in the (standard) string monoid $(\Sigma^*, \circ, \epsilon)$ such that $(a,b) \in I \Rightarrow ab \equiv ba$, for all $a, b \in \Sigma$ i.e., $w \equiv w'$, if there is a finite sequence of strings $w_1, \ldots, w_n$ s.t. $w_1 = w$, $w_n = w'$, and for each $i < n$, $w_i = uabv$, $w_{i+1} = ubav$, for some $(a,b) \in I$ and $u, v \in \Sigma^*$. Equivalence classes of $\equiv$ are called *traces* over (Σ, I). The trace generated by a string w is denoted by $[w]$. We use the following notations:

- $[\Sigma^*] = \{[w] \mid w \in \Sigma^*\}$, $[\Sigma^+] = \{[w] \mid w \in \Sigma^+\}$, and $[\Sigma] = \{[a] \mid a \in \Sigma\}$.

Concatenation of traces $[w], [v]$, denoted $[w][v]$, is defined as $[wv]$. For more deatails about traces see [17].

Now, let T be the set of all traces over (Σ, I). The *successor* relation $\rightarrow$ in T is defined as follows: $[w_1] \rightarrow [w_2]$ iff there is $a \in \Sigma$ such that $[w_1][a] = [w_2]$. The *prefix* relation $\leq$ in T is defined as a reflexive and

transitive closure of the successor relation i.e., $\leq = (\rightarrow)^*$. By $<$ we mean $\leq \Leftrightarrow id_T$. Let $\tau \in T$ and $Q \subseteq T$. We use the following notations:

- $\downarrow \tau = \{\tau' \in T \mid \tau' \leq \tau\}$, $\uparrow \tau = \{\tau' \in T \mid \tau \leq \tau'\}$,
- $\downarrow Q = \bigcup_{\tau \in Q} \{\tau' \in T \mid \tau' \leq \tau\}$, $\uparrow Q = \bigcup_{\tau \in Q} \{\tau' \in T \mid \tau \leq \tau'\}$.

We say that a subset Q of T *dominates* another subset R of T, if $R \subseteq \downarrow Q$. Two traces are *consistent*, if there is a trace in T dominating both of them and *inconsistent* otherwise. A set R of traces is said to be *proper*, if any two of its consistent traces are dominated by a trace in R, and *directed*, if arbitrary two traces in R are dominated by a trace in R. A set of traces Q is said to be *prefix-closed*, if $Q = \downarrow Q$.

By a *trace system* $\mathcal{T}$ over (Σ, I) we mean any prefix-closed and proper trace language over (Σ, I); traces $Tr(\mathcal{T})$ in a trace system $\mathcal{T}$ is called states of $\mathcal{T}$.

A trace together with its prefixes represents a partial execution of the system. Maximal (w.r.t. the inclusion ordering) directed subsets R of $Tr(\mathcal{T})$ is called *runs* of $\mathcal{T}$. A run represents a single maximal execution of the system $\mathcal{T}$. By a *path* in $R \subseteq Tr(\mathcal{T})$ we mean a maximal sequence $x = \tau_0 a_0 \tau_1 a_1 \ldots$ in R such that $\tau_i[a_i] = \tau_{i+1}$ for all $i \geq 0$. For convenience, we write also $x = \tau_0 \tau_1 \ldots$. Let $\downarrow x = \{\tau \in Tr(\mathcal{T}) \mid \tau \leq \tau_i,$ for $i \geq 0\}$ be a set of traces dominated by a path x. By an *observation* of a run R in $\mathcal{T}$ we mean any path x in R such that $R = \downarrow x$; we say also that x is an observation (of $\mathcal{T}$). Notice that an observation is such a path that is cofinal with some run. Thus, it carries the information about all actions executed in the run. A suffix $\tau_i a_i \tau_{i+1} a_{i+1} \ldots$ of an observation x is said to be an observation starting at τ_i.

Lemma 2.1 *A path $x = \tau_0 a_0 \tau_1 a_1 \ldots$ in $Tr(\mathcal{T})$ is an observation iff*

$$(\forall a \in \Sigma)(\forall i \in I\!N)(\exists j \geq i)(\tau_j[a] \notin Tr(\mathcal{T}) \text{ or } (a, a_j) \notin I).$$

Proof can be found in [13] or in [39].

Obviously, the above lemma holds for suffixes of observations. The comparison between the notion of an observation and the notion of a juste

computation can be found in [18]. In fact, each juste path is an observation, but not the other way round.

A subset $Q \subseteq Tr(\mathcal{T})$ is said to be *inevitable*, if each observation of $\mathcal{T}$ contains a state in Q. Inevitability is an important property for trace systems, discussed in [17,18].

Since one of the aims of this paper is to show a method of proving properties of trace systems by model checking, the finitely representable trace systems are of interest. They are called finite state trace systems.

Definition 2.1 *A trace system $\mathcal{T}$ is said to be* finite state*, if there is an equivalence relation $EQ \subseteq Tr(\mathcal{T}) \times Tr(\mathcal{T})$ in the set of traces of $\mathcal{T}$ satisfying the following two conditions:*

1. *EQ has a finite index,*
2. $(\forall \tau, \tau' \in Tr(\mathcal{T}))(\forall \alpha \in [\Sigma])$ $((\tau \ EQ \ \tau' \ and \ \tau\alpha \in Tr(\mathcal{T})) \Rightarrow (\tau\alpha \ EQ \ \tau'\alpha))$.

The above definition says that $\mathcal{T}$ contains a finite number of traces "distinguishable w.r.t. their continuations in $\mathcal{T}$". It should be noticed that $\mathcal{T}$ may have infinitely many traces with different histories i.e., $\downarrow \tau \neq \downarrow \tau'$, for infinitely many $\tau, \tau' \in Tr(\mathcal{T})$.

Examples of trace systems and of accompanying notions are given in the next section. Trace systems are commonly used for giving semantics to Elementary Net Systems [17] and finite state programs. In [13] trace systems were used for giving semantics to a wider class of concurrent programs, namely those for which equivalence classes are associated with first order interpretations. Below, EN-systems are defined and it is shown how to define their trace semantics. EN-systems serve us as examples of finite state concurrent systems.

3 EN-systems and trace semantics

Now, a subclass of Petri Nets, called EN-systems, is introduced.

Definition 3.1 (EN-system) *A* EN-system *is an ordered quadruple* $N = (B, E, F, c_0)$*, where* B *and* E *are finite, disjoint, nonempty sets of* places *and* transitions*, resp.,* $F \subseteq B \times E \cup E \times B$ *is the* flow *relation, with* $dom(F) \cup dom(F) = B \cup E$*, and* $c_0 \subseteq B$ *is the* initial case.

Any subset c of B is called a *case*. Nets are illustrated graphically using lines for representing transitions, circles for places, and arrows for the flow relation.

Definition 3.2 *Let* N *be a EN-system. For each* $x \in B \cup E$*, the following sets are defined:*

- $Pre(x) = \{y \mid (y, x) \in F\}$,
- $Post(x) = \{y \mid (x, y) \in F\}$,
- $Prox(x) = Pre(x) \cup Post(x)$.

Definition 3.3 (firing sequence) *We say that a transition* t *is* fireable *at a case* c *and leads to the case* c' *(written* $c[t> c'$*), if* $Pre(t) \subseteq c$*,* $Post(t) \subseteq c'$*, and* $c \Leftrightarrow Pre(t) = c' \Leftrightarrow Post(t)$.

A finite sequence of transitions $w = t_0 t_1 \ldots t_n$ *is said to be a* firing sequence *of* N*, if there is a sequence of cases* $c_1, \ldots, c_{n+1}$ *s.t.* $c_i[t_i > c_{i+1}$*, for* $i \leq n$*. This is denoted by* $c_0[w > c_{i+1}$*. The case* c_{n+1} *is said to be* reachable.

Definition 3.4 *A EN-system* N *is said to be* contact-free *iff for each reachable case* c *and for all* $t \in E$*, the following condition holds:*

- $Pre(t) \subseteq c$ *implies* $Post(t) \cap (c \Leftrightarrow Pre(t)) = \emptyset$.

Therefore, for contact-free nets a transiton t is fireable at c, if $Pre(t) \subseteq c$. All the EN-systems, used in the examples in the paper, are contact-free.

Next, it is shown how trace systems give semantics to EN-systems.

Definition 3.5 (trace semantics) *The trace system* $\mathcal{T}$ *over* (Σ, I) *represents behaviour of a EN-system* $N = (B, E, F, c_0)$*, if the following conditions hold:*

- $\Sigma = E$,
- $(a, b) \in I$ *iff* $Prox(a) \cap Prox(b) = \emptyset$, *and*
- $Tr(\mathcal{T}) = \{[w] \in [\Sigma^*] \mid w$ *is a firing sequence of transitions in* $N\}$.

Example 3.1 *Notice that trace systems giving a semantics to EN-systems are finite state. Let* $N = (B, E, F, c_0)$ *be a EN-system and* $\mathcal{T}$ *be the trace system representing behaviour of* N. $\mathcal{T}$ *is finite-state and EQ can be defined as follows:*

$$(\forall [w], [w'] \in Tr(\mathcal{T}))(\forall c \subseteq B)\ [w]\ EQ\ [w']\ \ iff\ \ (c_0[w > c \Leftrightarrow c_0[w' > c).$$

To allow the comparison of the approach presented here with that of Peled and Pnueli, all the examples shown below are taken from their paper [13].

Example 3.2 (inevitability) *Below, EN-system* N_1 *together with its trace system semantics* $\mathcal{T}_1$ *is presented.*

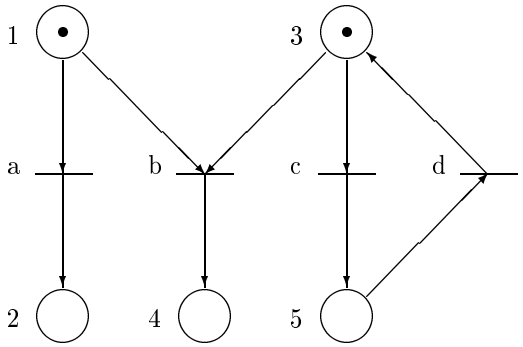


Figure 1: EN-system N_1

The EN-system $N_1 = (B_1, E_1, F_1, c_0^1)$, where

- $B_1 = \{1, 2, 3, 4, 5\}$,
- $E_1 = \{a, b, c, d\}$,
- $F_1 = \{(1, a), (a, 2), (1, b), (b, 4), (3, b), (3, c), (c, 5), (5, d), (d, 3)\}$,
- $c_0^1 = \{1, 3\}$.

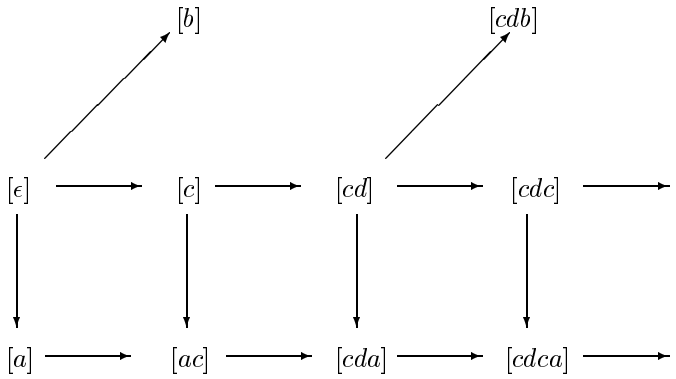


Figure 2: Trace Semantics $\mathcal{T}_1$ of System N_1

The independence alphabet (Σ, I) is defined as follows:

- $\Sigma = \{a, b, c, d\}$,
- $I = \{(a, c), (c, a), (a, d), (d, a)\}$.

There are infinitely many finite runs R_i and one infinite run R in the trace system $\mathcal{T}_1$:

- $R_i = \downarrow [(cd)^i b]$, for $i \geq 0$,
- $R = \bigcup_{i=1}^{\infty} \downarrow [(cd)^i a]$

Every path in $Tr(\mathcal{T}_1)$ except for $x = [\epsilon][c][cd][cdc]...$ is an observation of $\mathcal{T}_1$.

The following is an example of an inevitability property:

- INEVITABILITY: Either a or b will be eventually executed.
  (i.e., cases containing 2 or 4 will be reached inevitably).

One could be estonished that either a or b is inevitable as there is an infinite path $x = [\epsilon][c][cd][cdc]\ldots$ in which neither a nor b is executed. But this path is not an observation and according to the definition, a property is inevitable if it holds for each observation.

Example 3.3 (serializability) *Below, EN-system N_2 and its trace semantics is presented. Then, an example of a serializability property is given.*

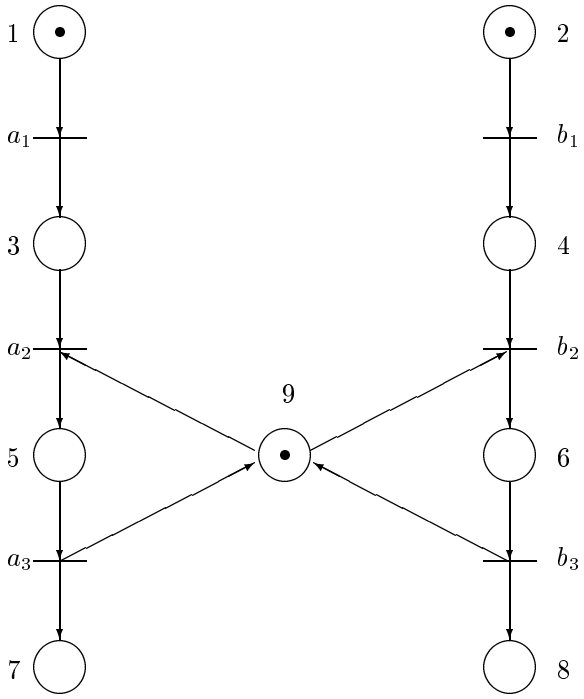


Figure 3: EN-system N_2

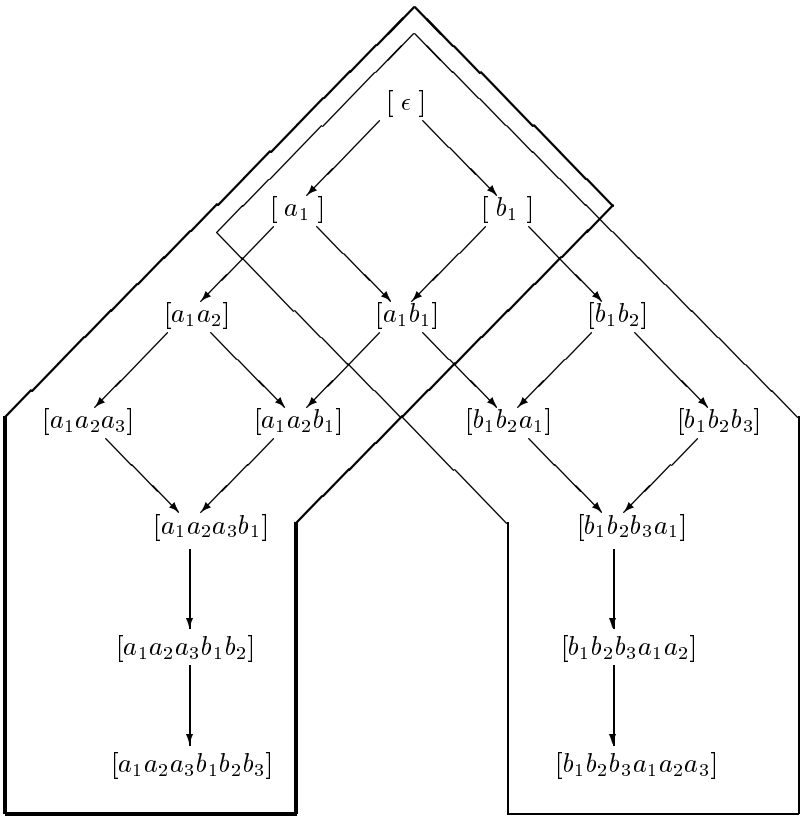


Figure 4: Trace Semantics $\mathcal{T}_2$ of N_2

The EN-system $N_2 = (B_2, E_2, F_2, c_0^2)$ is defined as follows:

- $B_2 = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$,
- $E_2 = \{a_1, a_2, a_2, b_1, b_2, b_3\}$,
- $F_2 = \{(1, a_1), (a_1, 3), (3, a_2), (a_2, 5), (5, a_3), (a_3, 7), (a_3, 9), (9, a_2),$
  $(9, b_2), (2, b_1), (b_1, 4), (4, b_2), (b_2, 6), (6, b_3), (b_3, 9), (b_3, 8)\}$,

- $c_0^2 = \{1, 2, 9\}$.

The trace system $\mathcal{T}_2$ contains two runs, marked by the thin and the thick lines.

- SERIALIZABILITY:

The net N_2 can be thought of as implementing a database with two transactions $T_1 = \{a_1, a_2, a_3\}$ and $T_2 = \{b_1, b_2, b_3\}$. Each transition can represent database read or write operation. An implementation is correct, if transactions are serializable. Serializability of T_1 and T_2 [13] in terms of trace semantics is as follows: In every run, there exists an observation in which either all the operations of T_1 appear before those of T_2, or the other way round, i.e., there is an observation, which behaves in a serial manner. One can check that indeed in each run there is an observation satisfying the above requirement, namely:

- $[\epsilon][a_1][a_1a_2][a_1a_2a_3][a_1a_2a_3b_1][a_1a_2a_3b_1b_2][a_1a_2a_3b_1b_2b_3]$ in the "thick" line marked run, and
- $[\epsilon][b_1][b_1b_2][b_1b_2b_3][b_1b_2b_3a_1][b_1b_2b_3a_1a_2][b_1b_2b_3a_1a_2a_3]$ in the "thin" line marked run.

In the next part of the paper it is shown how to express the mentioned properties in the logic and then how to prove them by model checking.

4 Logic CTL_P and its semantics

Next, the language of CTL_P is defined and then semantics is presented.

4.1 Syntax of CTL_P

Let Σ be a fixed set of action names. Now, we define the set of state and path formulas of CTL_P.

Let PV be a set of propositional variables.

1. Each $p \in PV$ is a state formula,

2. if φ and ψ are state formulas, then so are $\varphi \wedge \psi$, $\neg\varphi$,
3. if φ is a state formula, then so are $Y_a\varphi$ (for $a \in \Sigma$), and $H\varphi$,
4. if φ is a state formula, then $X_a\varphi$ (for $a \in \Sigma$), $G\varphi$, and $\varphi U\psi$ are path formulas,
5. if φ is a path formula, then $E\varphi$ is a state formula.

The symbol E can be called an observation quantifier (it corresponds to path quantifiers in CTL). The other symbols have the following intuitive meaning: X_a - next step "along" a, U - Until, G - always in the future, Y_a - backward step "along" a, H - always in the past. Path formulas are interpreted over observations and state formulas are evaluated at states of models. Formulas of the form $Y_a\varphi$, $H\varphi$ are called past formulas.

The following abbreviations are used:

- $\varphi \vee \psi \stackrel{def}{=} \neg(\neg\varphi \wedge \neg\psi)$,
- $true \stackrel{def}{=} \varphi \vee \neg\varphi$, for any φ,
- $\varphi \Rightarrow \psi \stackrel{def}{=} \neg\varphi \vee \psi$,
- $F\varphi \stackrel{def}{=} trueU\varphi$,
- $P\varphi \stackrel{def}{=} \neg H\neg\varphi$,
- $EX\varphi \stackrel{def}{=} \bigvee_{a\in\Sigma} EX_a\varphi$, $AX\varphi \stackrel{def}{=} \neg EX\neg\varphi$,
- $Y\varphi \stackrel{def}{=} \bigvee_{a\in\Sigma} Y_a\varphi$,
- $AG\varphi \stackrel{def}{=} \neg EF\neg\varphi$,
- $A(\varphi U\psi) \stackrel{def}{=} \neg(E(\neg\psi U(\neg\varphi \wedge \neg\psi)) \vee EG(\neg\psi))$.

The lack of symmetry between backward and forward operators stems from the fact that the past of a state is conflict-free whereas the future may not.

In this paper we also consider a logic CTL_{P-}, which is a restriction of CTL_P s.t. past formulas cannot be nested.

4.2 Semantics of CTL$_P$

Now, we define formally the semantics of CTL$_P$.

Definition 4.1 *A* frame *(for a trace system $\mathcal{T}$ over (Σ, I)) is a triple $F_{v_0} = (W, \rightarrow, v_0)$, where $W = Tr(\mathcal{T})$ is the set of traces of the trace system $\mathcal{T}$, $\rightarrow \subseteq W \times \Sigma \times W$ is a labelled transition relation s.t. $\tau \stackrel{a}{\rightarrow} \tau'$ iff $\tau' = \tau[a]$, and $v_0 = [\epsilon]$.*

Definition 4.2 *A* model *(for a trace system $\mathcal{T}$ over (Σ, I)) is an ordered pair $M = (F_{v_0}, V)$, where $F_{v_0} = (W, \rightarrow, v_0)$ is a frame for $\mathcal{T}$ and $V : W \Leftrightarrow 2^{PV}$ is a valuation function.*

The notion of *truth* in M is defined by the relation $\models$ as follows:

1. $M, w_0 \models p$ iff $p \in V(w_0)$, for $p \in PV$,
2. if φ, ψ are state formulas,

   $M, w_0 \models \neg\varphi$ iff not $M, w_0 \models \varphi$,

   $M, w_0 \models \varphi \wedge \psi$ iff $M, w_0 \models \varphi$ and $M, w_0 \models \psi$,
3. $M, w_0 \models H\varphi$ iff $M, w' \models \varphi$, for all $w' : w' < w_0$,

   $M, w_0 \models Y_a\varphi$ iff $M, w' \models \varphi$, for some $w' : w' \stackrel{a}{\rightarrow} w_0$,
4. if $x = w_0 a_0 w_1 a_1 \ldots$ is an observation starting at w_0, then

   $M, x \models X_a\varphi$ iff $M, w_1 \models \varphi$ and $a_0 = a$,

   $M, x \models G\varphi$ iff $M, w_i \models \varphi$ for all $i \geq 0$,

   $M, x \models \varphi U \psi$ iff there is $k \geq 0$ s.t. $M, w_k \models \psi$, and for all $0 \leq i < k$: $M, w_i \models \varphi$,
5. if φ is a path formula, then

   $M, w_0 \models E\varphi$ iff $M, x \models \varphi$, for some observation x starting at w_0.

We say that a formula φ is *valid in a model* M (written $M \models \varphi$), if $M, v_0 \models \varphi$. Such notion of validity in model is sometimes called *the anchored validity*. A formula φ is said to be *valid*, if $M \models \varphi$, for all models M.

The language of CTL_P contains all the CTL formulas (with slightly different semantics, tuned to observations) and moreover the formulas with the past modalities H and Y_a.

Since our logic needs to be able to speak about actions, we have defined labelled next and backward step operators. However, we have done that only for simplicity. To show that we could have avoided introducing labelled operators we give the following example of a valuation function, which encodes labels of transitions. Then, labelled operators are expressible using their unlabelled versions.

Example 4.1 *Let* $PV_\Sigma = \{p_a \mid a \in \Sigma\} \subseteq PV$ *and let* V *satisfy the following condition:*

- $p_a \in V(\tau)$ *iff* $\sharp_a(\tau)$ *is odd,* ($\sharp_a(\tau)$ *is the number of occurrences of* a *in* τ*).*

Then, $EX_a\varphi$ *would be equivalent to a formula* $(p_a \Rightarrow EX(\neg p_a \wedge \varphi)) \wedge (\neg p_a \Rightarrow EX(p_a \wedge \varphi))$ *and* $Y_a\varphi$ *would be equivalent to a formula* $(p_a \Rightarrow Y(\neg p_a \wedge \varphi)) \wedge (\neg p_a \Rightarrow Y(p_a \wedge \varphi))$.

4.3 Expressiveness of CTL_P

We give several examples of expressiveness of CTL_P. Then, we convince the reader that for finite state models, if we can express properties of partial executions and runs, then we can specify serializability. Let $M = (F_{v_0}, V)$ be a model for a trace system $\mathcal{T}$.

- $M, v_0 \models AG\varphi - \varphi$ is an invariant in $\mathcal{T}$,
- $M, v_0 \models AF\varphi - \varphi$ is inevitable in $\mathcal{T}$, (under a concurrency fairness assumption),
- $M, v_0 \models EF\varphi - \varphi$ is possible in $\mathcal{T}$,

- $M, v_0 \models EF(H\varphi \wedge \varphi)$ – there is a partial execution in $\mathcal{T}$ s.t. φ holds at all its states,
- $M, v_0 \models EG(H\varphi \wedge \varphi)$ – there is a run in $\mathcal{T}$ s.t. φ holds at all its states,
- $M, v_0 \models AF(P\varphi \vee \varphi)$ – φ holds at some state of each run in $\mathcal{T}$,
- $M, v_0 \models EF(Y_a(true) \wedge Y_b(true))$ – actions a and b are independent in $\mathcal{T}$,
- $M, v_0 \models AG(\varphi \Rightarrow P\psi)$ – always if φ holds, ψ held in the past; this formula allows for specifying snapshots of concurrent programs [13, 16], (φ and ψ do not contain temporal formulas).

Assume now that the trace system $\mathcal{T}$ is finite state and given an equivalence relation EQ in $Tr(\mathcal{T})$ s.t. $|EQ| = n$. Moreover, we require that a valuation function of equivalent states is the same i.e., $V(\tau) = V(\tau')$, if $\tau \; EQ \; \tau'$. Now, we show how serializability of two transactions T_1 and T_2 can be specified.

Obviously, we consider here a simplified version of serializability without taking into account aborts and any consistency conditions of specific operations of transactions. All these requirements could be also specified by translating first order formulas, defined in [16], into their propositional versions. Here, we concentrate our attention on specifying that transactions can be serialized.

Let $M = (F_{v_0}, V)$ be a model for $\mathcal{T}$. Firstly, assume that transactions are executed only once. Denote by $before_i$ and $after_i$ the assertions meaning that control is before or after the execution of transaction T_i, respectively. Then, serializability of T_1 and T_2 can be expressed as follows:

1. $(before_1 \wedge before_2) \Rightarrow AFP((before_1 \wedge after_2) \vee (before_2 \wedge after_1))$,
2. $AG(((before_1 \wedge after_2) \vee (before_2 \wedge after_1)) \Rightarrow AF(P(after_1 \wedge after_2) \vee (after_1 \wedge after_2)))$.

The first formula expresses that each run contains a state at which either control is before the execution of T_1 and after the execution of T_2, or the other way round. The second formula says that each run contains a state at which control is after the execution of T_1 and T_2.

Secondly, we assume that T_1 and T_2 are executed infinitely many times. We denote by "$quiescent_i$" and "$active_i$" the assertions meaning that a transaction T_i has already terminated or has not yet started, and that T_i is active, respectively. Moreover, we assume that we have a set of assertions $\{\theta_1, \ldots, \theta_n\}$, identyfing uniquely states belonging to different equivalence classes of EQ i.e., at each state exactly one θ_i holds and at states belonging to the same equivalence class the same θ_j holds. Then, serializability of T_1 and T_2 can be expressed as follows:

SER1: $AG[\bigwedge_{j=1}^{n}((\theta_j \wedge \bigwedge_{i=1}^{2} quiescent_i) \Rightarrow AF(\bigwedge_{i=1}^{2} P(active_i \wedge P(\theta_j)))]$

SER2: $AG[(\bigwedge_{j=1}^{n}((\theta_j \wedge \bigvee_{i=1}^{2} active_i) \Rightarrow AFP((\bigwedge_{i=1}^{2} quiescent_i \wedge P(\theta_j)))]$

SER3: $AG[\bigwedge_{j=1}^{n}((\theta_j \wedge \bigwedge_{i=1}^{2} quiescent_i) \Rightarrow$
$AG((\bigwedge_{i=1}^{2}(quiescent_i \wedge P(active_i \wedge P(\theta_j))) \Rightarrow P(\bigwedge_{i=1}^{2} quiescent_i \wedge$
$P(\theta_j))))]$

The formula SER1 expresses that if at some state s transactions T_1 and T_2 are quiescent, then for each run in the future of s transactions T_1 and T_2 will become active. The formula $P(\theta_j)$ is used in order to guarantee that the transactions become active in the future of s. The formula SER2 says that if at some state s the transactions T_1 and T_2 are active, then for each run at some state in the future of s the transactions T_1 and T_2 will become quiescent. We have to add also the third formula. Informally saying, the formula SER3 expresses that for each trace "starting" and "ending" at a state, where T_1 and T_2 are quiescent, but were active at some its prefix, there is a linearisation, which behaves in a serial manner.

Example 4.2 *Now, it is shown how the properties, we have discussed in the former examples, can be formally expressed by CTL_P formulas. Firstly, models are defined. Let $PV_i = \{p_b \mid b \in B_i\}$, where B_i is the set of places of EN-system N_i, for $i \in \{1, 2\}$. The valuation function assigns to each state these propositions which correspond to marked places.*

The model $M_1 = (F^1_{[\epsilon]}, V_1)$, for the trace system $\mathcal{T}_1$ is defined as follows:

- $F^1_{[\epsilon]} = (W_1, \rightarrow_1, [\epsilon])$ is the frame for $\mathcal{T}_1$,
- $V_1 : W_1 \Leftrightarrow\!\!\rightarrow 2^{PV_1}$ s.t.
  - $p_b \in V_1([w])$ iff $p_b \in c$, for the case c s.t. $c^1_0[w > c$ in N_1.

In the following way, the property, discussed in Example 3.2, can be expressed by CTL_P formulas:

- INEVITABILITY: $p_1 \wedge p_3 \Rightarrow AF(p_2 \vee p_4)$,

The model $M_2 = (F^2_{[\epsilon]}, V_2)$ for the trace system $\mathcal{T}_2$ is defined as follows:

- $F^2_{[\epsilon]} = (W_2, \rightarrow_2, [\epsilon])$ is the frame for $\mathcal{T}_2$,
- $V_2 : W_2 \Leftrightarrow\!\!\rightarrow 2^{PV_2}$ s.t.
  - $p_b \in V_2([w])$ iff $b \in c$, for the case c s.t. $c^2_0[w > c$ in N_2.

In this case T_1 and T_2 can occur only once. Thus, our specification is simpler. In terms of propositional variables, the property can be expressed as:

- SERIALIZABILITY:

1. $p_1 \wedge p_2 \wedge p_9 \Rightarrow AFP((p_7 \wedge p_2 \wedge p_9) \vee (p_1 \wedge p_8 \wedge p_9))$,
2. $AG[((p_7 \wedge p_2 \wedge p_9) \vee (p_1 \wedge p_8 \wedge p_9)) \Rightarrow AF(P(p_7 \wedge p_8 \wedge p_9) \vee (p_7 \wedge p_8 \wedge p_9))]$.

In the following sections, it is shown how to prove the above property by model checking.

5 Acceptors for finite state trace systems and model generators

Firstly, we define formally a notion of a finite state model for CTL_P.

Definition 5.1 *A model $M = (F_{v_0}, V)$ (for a trace system $\mathcal{T}$) is said to be* finite state, *if there is an equivalence relation $EQ \subseteq Tr(\mathcal{T}) \times Tr(\mathcal{T})$ in the set of traces of $\mathcal{T}$ satisfying the conditions of the Definition 2.1 and*

3. *for each $\tau, \tau' \in Tr(\mathcal{T})$, if τ EQ τ', then $V(\tau) = V(\tau')$ (EQ* respects *V).*

Then, we show how to represent finite state models in a finite way. First, we show how to represent frames by a special kind of labelled transition systems.

Now, the definitions of labelled transition systems and concurrent transition systems are given, and related to trace systems. Concurrent transition systems have been introduced in [40] and investigated by many authors (see [41, 42]).

Definition 5.2 (labelled transition system) *A* labelled transition system (lts) *is a triple*
$F = (W, \Sigma, \rightarrow)$ such that:

1. *W is a non-empty, countable set of states,*
2. *Σ is a non-empty, countable set of actions,*
3. *$\rightarrow \subseteq W \times \Sigma \times W$ is a transition relation; we write $w \stackrel{a}{\rightarrow} w'$ instead of $(w, a, w') \in \rightarrow$,*

The following notation is also used:

- $w \rightarrow w'$, if there is $a \in \Sigma$ s.t. $w \stackrel{a}{\rightarrow} w'$,
- $w \rightarrow^* w'$, it there is a sequence of states $w_0, \ldots, w_n$ s.t. $w = w_0$, $w' = w_n$, and $w_i \rightarrow w_{i+1}$ for $0 \leq i < n$.

Definition 5.3 (concurrent transition system) *A* concurrent transition system (cts) *is a 4-tuple*
$F = (W, \Sigma, \rightarrow, I)$ such that:

1. *$(W, \Sigma, \rightarrow)$ is a labelled transition system,*

2. (Σ, I) *is an independence alphabet,*

3. $(\forall w, w', w'' \in W)(\forall a, b \in \Sigma)$: $(w \xrightarrow{a} w' \xrightarrow{b} w''$ *and* $(a,b) \in I)$ *implies* $\exists w''' : w \xrightarrow{b} w''' \xrightarrow{a} w''$,

4. $(\forall w, w', w'' \in W)(\forall a, b \in \Sigma)$: $(w \xrightarrow{a} w'$ *and* $w \xrightarrow{b} w''$ *and* $(a,b) \in I)$ *implies* $\exists w''' : w' \xrightarrow{b} w'''$ *and* $w'' \xrightarrow{a} w'''$,

5. $(\forall w, w', w'' \in W)(\forall a \in \Sigma)$: $(w \xrightarrow{a} w'$ *and* $w \xrightarrow{a} w'')$ *implies* $w' = w''$.

Condition 3) requires that independent actions can commute, condition 4) is called a "diamond property". Condition 5) expresses unambiguity. In our case, this condition does not introduce any limitation, but it is convenient for using sequences of actions to denote paths.

Definition 5.4 *Let* $F = (W, \Sigma, \rightarrow, I)$ *be a concurrent transition system and* $w_0 \in W$.

- $F_{w_0} = (W, \Sigma, \rightarrow, I, w_0)$ *is said to be a* rooted concurrent transition system - rcts, *for short.*

- *A* path *in* F_{w_0} *is any finite or infinite sequence* $x = w_0 a_0 w_1 a_1 \ldots$ *s.t.* $w_i \xrightarrow{a_i} w_{i+1}$, *for* $i \geq 0$.

- $en(w_0) = \{a \in \Sigma \mid (\exists w \in W) : w_0 \xrightarrow{a} w\}$ *is the set of all the actions enabled at* w_0.

The following notations are used. Let $x = w_0 a_0 w_1 a_1 \ldots$ be a path in F_{w_0}.

- $x/\Sigma = a_0 a_1 \ldots$,

- $Tr(F_{w_0}) = \{[x/\Sigma] \mid x$ is a finite path in $F_{w_0}\}$,

- F_{w_0} is said to *accept* a trace system $\mathcal{T}$ over (Σ, I), if $Tr(\mathcal{T}) = Tr(F_{w_0})$.

- A path x is said to be an *observation* in F_{w_0}, if $[\epsilon][a_0][a_0a_1]\ldots$ is an observation in $Tr(F_{w_0})$.
- A suffix $w_ia_iw_{i+1}a_{i+1}\ldots$ of an observation x is said to be an observation starting at w_i.

Next, it is shown that rooted concurrent transition systems accept trace systems.

Fact 5.1 *Let F_{w_0} be a rooted concurrent transition system. Then, $Tr(F_{w_0})$ over (Σ, I) is a trace system.*

Proof. It is easy to show that condition 4 in the definition of *cts* implies that $Tr(F_{w_0})$ is proper and condition 3 guarantees that $Tr(F_{w_0})$ is prefix-closed.

Moreover, all trace systems have got their acceptors.

Fact 5.2 *Let $\mathcal{T}$ be any trace system. Then, there exists a rooted concurrent transition system F_{w_0} s.t. $Tr(F_{w_0}) = Tr(\mathcal{T})$.*

Proof. Clearly, $F_{w_0} = (Tr(\mathcal{T}), \Sigma, \rightarrow, I, [\epsilon])$ is a rooted concurrent transition system, accepting $\mathcal{T}$, where $\tau \stackrel{a}{\rightarrow} \tau'$ iff $\tau' = \tau[a]$, for $a \in \Sigma$.

If a trace system $\mathcal{T}$ is finite state and EQ is an equivalence relation in $Tr(\mathcal{T})$, as described in the Definition 2.1, then there exists a finite rooted concurrent transition system accepting $\mathcal{T}$.

Definition 5.5 *The quotient structure of $Tr(\mathcal{T})$ by EQ is a 5-tuple $F_{w_0} = (W, \Sigma, \rightarrow, I, w_0)$, where:*

- *$W = \{[\tau]_{EQ} \mid \tau \in Tr(\mathcal{T})\}$ is a set of states,*
- *(Σ, I) is an independence alphabet,*
- *$\rightarrow \subseteq W \times \Sigma \times W$ is a transition relation such that $[\tau]_{EQ} \stackrel{a}{\rightarrow} [\tau']_{EQ}$, if there are traces $\tau_1 \in [\tau]_{EQ}, \tau_1' \in [\tau']_{EQ}$, and $a \in \Sigma$ s.t. $\tau_1[a] = \tau_1'$,*

- $w_0 = [\epsilon]_{EQ}$.

It is easy to check that F_{w_0} is a rooted concurrent transition system accepting $\mathcal{T}$. Therefore, if a trace system $\mathcal{T}$ and an equivalence relation EQ in $Tr(\mathcal{T})$ is given, then Definition 5.5 shows how to define the finite rcts accepting $\mathcal{T}$.

Example 5.1 (acceptors for trace systems) *Below, we show the definitions of finite acceptors for the trace systems $\mathcal{T}_1$ and $\mathcal{T}_2$ of the Examples 3.2 and 3.3.*

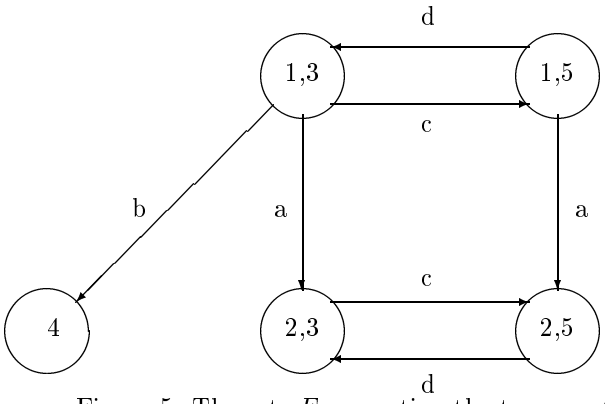


Figure 5: The rcts F_1 accepting the trace system $\mathcal{T}_1$

The rcts $F^1_{w^1_0} = (W_1, \Sigma_1, \rightarrow_1, I_1, w^1_0)$ accepting the trace system $\mathcal{T}_1$ is defined as follows:

- $W_1 = \{\{1,3\}, \{1,5\}, \{4\}, \{2,3\}, \{2,5\}\}$,
- $\Sigma_1 = \{a, b, c, d\}$,
- $\rightarrow_1 = \{(\{1,3\}, b, \{4\}), (\{1,3\}, a, \{2,3\}), (\{1,3\}, c, \{1,5\}),$
  $(\{1,5\}, d, \{1,3\}), (\{1,5\}, a, \{2,5\}),$
  $(\{2,5\}, d, \{2,3\}), (\{2,3\}, c, \{2,5\})\}$,
- $I_1 = \{(a,c), (c,a), (d,a), (a,d)\}$,

- $w_0^1 = \{1, 3\}$.

One can easily notice that, in this case, $F^1_{w_0^1}$ is the sequential case graph of N_1, extended by the dependence relation. The trace system $\mathcal{T}_2$ itself defines the finite rcts $F^2_{w_0^2} = (W_2, \Sigma_2, \rightarrow_2, I_2, w_0^2)$, where:

- $W_2 = Tr(\mathcal{T}_2)$,
- $\Sigma_2 = \{a_1, a_2, a_3, b_1, b_2, b_3\}$,
- $\tau \stackrel{a}{\rightarrow}_2 \tau'$ iff $\tau[a] = \tau'$,
- $I_2 = \Sigma_2 \times \Sigma_2 \Leftrightarrow (\{a_2, a_3, b_2, b_3\}^2 \cup$
  $\{(a_1, a_1), (b_1, b_1), (a_1, a_2), (a_2, a_1), (b_1, b_2), (b_2, b_1)\})$,
- $w_0^2 = [\epsilon]$.

Now, let $F_w = (W, \Sigma, \rightarrow, I, w)$ be a *rcts*. Define an *acceptance function* $AC : Tr(F_w) \Leftrightarrow\!\!\rightarrow W$ assigning to each trace from $Tr(F_w)$ the state of F_w which accepts this trace, i.e., $AC(\tau) = w'$ iff there is a finite path x from w to w' s.t. $[x/\Sigma] = \tau$. Thanks to the conditions 4) and 5) in Definition 5.3, AC is defined correctly. In the standard way AC is extended on subsets of $Tr(F_w)$: $AC(P) = \{AC(\tau) \mid \tau \in P\}$, for $P \subseteq Tr(F_w)$.

Now, we define the notion of a model generator for a finite state model for a trace system $\mathcal{T}$.

Definition 5.6 *Let $M = (F_{v_0}, V)$ be a finite state model for a trace system $\mathcal{T}$, and let EQ be an equivalence relation in $Tr(\mathcal{T})$ respecting V. Then, a model generator for M is defined as $M_G = (F_{w_0}, V_G)$, where F_{w_0} is the quotient structure of $Tr(\mathcal{T})$ by EQ and $V_G : [Tr(\mathcal{T})]_{EQ} \Leftrightarrow\!\!\rightarrow 2^{PV}$ s.t. $V_G([\tau]_{EQ}) = V(\tau)$, for any $\tau \in Tr(\mathcal{T})$.*

We say that a formula φ is true in a model generator M_G at w_0 (written $M_G, w_0 \models \varphi$), if $M, v_0 \models \varphi$.

Before discussing a model checking algorithm for CTL_P, we show that, in fact, we can restrict ourselves to give an algorithm for CTL_{P-} - the restriction of CTL_P without nested past modalities and still we can

check all the properties we are interested in. The only property, we have discussed, which required nested past operators, was serializability. But we show that for checking serializability, we can use a model checking algorithm for CTL_{P-}. This follows from the following observation. Let M_τ be a restriction of the model M to states $\uparrow \tau \cap Tr(\mathcal{T})$ i.e., M_τ is a submodel of M, generated by τ. Let θ be a proposition which holds only at the beginning state of each model M_τ i.e., $\theta \equiv Hfalse$. Notice that:

- $M \models SER1 \wedge SER2 \wedge SER3$ iff for each $\tau \in Tr(\mathcal{T})$: $M_\tau \models SER1' \wedge SER2' \wedge SER3'$, where

$SER1'$: $(\bigwedge_{i=1}^{2} quiescent_i) \Rightarrow AF(\bigwedge_{i=1}^{2} P(active_i))$

$SER2'$: $(\bigvee_{i=1}^{2} active_i) \Rightarrow AFP(\bigwedge_{i=1}^{2} quiescent_i)$

$SER3'$: $(\bigwedge_{i=1}^{2} quiescent_i) \Rightarrow AG((\bigwedge_{i=1}^{2}(quiescent_i \wedge P(active_i)) \Rightarrow P(\bigwedge_{i=1}^{2} quiescent_i \wedge \neg\theta))$

Since M is a finite state model, then there are at most $|EQ|$ non-isomorphic submodels M_τ. Let $M_G = (F_{w_0}, V_G)$, where $F_{w_0} = (W, \Sigma, \rightarrow, I, w_0)$ be a model generator for M and let $M_G^w = (F_w, V_G^w)$ be a model generator for M_τ, where $AC(\tau) = w \in W$. Consequently, $M \models SER1 \wedge SER2 \wedge SER3$ iff $M_G^w, w \models SER1' \wedge SER2' \wedge SER3'$, for each $w \in W$.

Therefore, fortunately we can give a model checking algorithm for CTL_{P-} and we can still verify all the properties, discussed so far.

Unfortunately, as we show in the next section, model checking for CTL_{P-} is NP-hard.

6 NP-hardness of CTL_P model checking

In this section we show that model checking for CTL_{P-}, and therefore for CTL_P is NP-hard. Our problem has the following formulation: given a model generator $M_G = (F_{w_0}, V_G)$ and a formula φ, is φ true in M_G at w_0 ?

We prove that 3-SAT problem is polynomialy reducible to determining whether $M_G, w_0 \models \varphi$, for some CTL_P formula φ.

Let $\psi = c_1 \wedge \ldots \wedge c_m$ be a boolean formula in 3-CNF, where $c_i = l_{i_1} \vee l_{i_2} \vee l_{i_3}$, for $1 \leq i \leq m$, $l_{i_j} = x_k$ or $\neg x_k$ for some k s.t. $1 \leq k \leq n$, $x_1, \ldots, x_n$ are the propositional variables appearing in ψ.

Let $M_G = (F_{w_0}, V_G)$ be a model generator (see the figure below), defined as follows:

- let $X = \{x_i \mid 1 \leq i \leq n\} \cup \{x'_i \mid 1 \leq i \leq n\}$, $Y = \{y_i \mid 0 \leq i \leq n\}$,
- $F_{w_0} = (W, \Sigma, \rightarrow, I, w_0)$, where
- $W = X \cup Y$, $\Sigma = X \times Y \cup Y \times X$, $I = \emptyset$,
- $\rightarrow = \{(y_{i-1}, (y_{i-1}, x_i), x_i), (y_{i-1}, (y_{i-1}, x'_i), x'_i),$
  $(x_i, (x_i, y_i), y_i), (x'_i, (x'_i, y_i), y_i) \mid 1 \leq i \leq n\}$,
- $w_0 = y_0$,
- $PV = \{c_i \mid 1 \leq i \leq m\}$,
- $V(x_i) = \{c_j \mid x_i$ appears as a literal in $c_j\}$,
- $V(x'_i) = \{c_j \mid \neg x_i$ appears as a literal in $c_j\}$,
- $V(y_i) = \emptyset$, for $0 \leq i \leq n$.

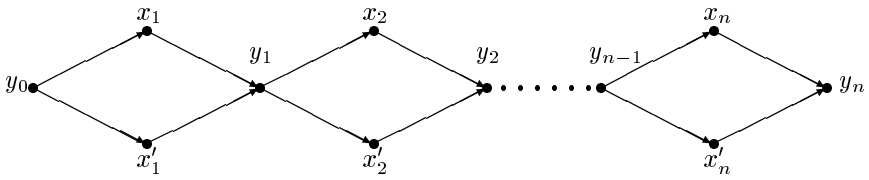


Figure 8: The structure F_{w_0} of the model generator M_G

The following equivalence holds:

*) ψ is satisfiable iff $M_G, w_0 \models EF(Pc_1 \wedge \ldots \wedge Pc_m)$.

The above follows from the fact that since the dependence relation $I = \emptyset$, therefore the model, generated by M_G, is a tree. Therefore, all c_i must be true somewhere at the same path, which represents one of the possible valuations of propositional variables x_i. This reduction is polynomial. Hence, model checking for CTL_P is NP-hard.

In fact we have proved even more. Namely, that model checking is NP-hard, even if past modalities cannot be nested, i.e., model checking for CTL_{P-} is NP-hard.

The next part of this paper is devoted for defining a model checking algorithm for CTL_{P-} and then, it is shown how our method can be extended to cover the whole language of CTL_P.

7 Model checking for $\mathrm{CTL}_{P\Leftrightarrow}$

Altough, CTL_{P-} seems to be a very mild extension of CTL, it turns out that using a model checking algorithm for CTL_{P-} we can check all the partial order properties, we have discussed so far. Therefore, a model checking method for CTL_{P-} is of interest. Because of NP-hardness, we can hardly believe in a polynomial time model checking algorithm. Thus, the best we can offer is an algorithm of one exponential complexity. Moreover, as we show later, as soon as we know how to check CTL_{P-} formulas we can apply this method to the whole language of CTL_P and its extension to CTL_P^*.

Unfortunately, any of the known methods for model checking, applied to CTL or CTL*, cannot be used in the case of CTL_{P-}. Showing why all these methods fail is going to be a good introduction to our method of model checking.

Let $M_G = (F_{w_0}, V_G)$ be a model generator for M (we assume that the set PV of propositional variables is finite) and ψ be a CTL_{P-} formula to be checked over. Firstly, let's try to apply a method, used for CTL [21]. Immediately we find an obstacle. We do not know how to label states of a model generator M_G with subformulas of ψ containing past operators. This problem stems from the fact that it may happen that for some $\tau, \tau' \in Tr(F_{w_0})$: $AC(\tau) = AC(\tau') = w$ in M_G and for some $H\varphi \in Subformulas(\psi)$, $M, \tau \models H\varphi$ and $M, \tau' \models \neg H\varphi$.

Secondly, let's try to apply a method, used for LTTL [19] and CTL* [24]. Define a cross product of M_G with all subsets of $Subformulas(\psi) \cup PV$. Then, we can obtain a structure which is not a concurrent transition system. In fact, it can violate all the conditions, but that, which cannot be repaired is a diamond property. It may happen that for some trace τ, two different states w and w' in the new structure accept it. Since, w and w' may have been assigned different formulas, therefore our construction fails.

Looking at automata theoretic constructions [20] we realize that the same problems appear as in the former case. Therefore, we need a new technique. The general idea is to unwind M_G s.t. we can assign past formulas to states of a new structure and then use the most efficient method of CTL model checking, tunned to observations.

Let $F_{w_0} = (W, \Sigma, \rightarrow, I, w_0)$ be a rcts, $w \in W$, $W' \subseteq W$, and $\tau \in [\Sigma^*]$. The following notations are used:

- $\Sigma(\tau) = \{a \in \Sigma \mid \tau = \tau_1[a]\tau_2$, for some $\tau_1, \tau_2 \in [\Sigma^*]\}$ is the set of action names "occurring" in the trace τ.
- $en(w, W') = \{a \in \Sigma \mid (\exists w' \in W') : \ w \stackrel{a}{\rightarrow} w'\}$ defines actions, enabled at w and leading to the states from W',
- $\Sigma_I = \{a \in \Sigma \mid (\exists b \in \Sigma)\ (a, b) \in I\}$ is the set of actions for wich there is at least one independent action,
- $I(\Sigma) = \{\Sigma' \subseteq \Sigma \mid (\exists a \in \Sigma) : \ \{a\} \times \Sigma' \subseteq I\} \cup \{\Sigma\}$ is the family of sets of actions Σ' s.t. there is at least one action independent with all its elements or $\Sigma' = \Sigma$.

We need the following lemmas.

Lemma 7.1 *A maximal path $x = w_0 a_0 w_1 a_1 \ldots$ is an observation in F_{w_0} iff*

$$(\forall a \in \Sigma)(\forall i \in I\!N)(\exists j \geq i)(a \notin en(w_j or\ ((a, a_j) \notin I))$$

Proof follows directly from Lemma 2.1.

Obviously, the above lemma holds also for observations starting at any $w \in W$.

The next lemma is a special case of Levi's Lemma for traces [17]. We present it with a short proof.

Lemma 7.2 *For each* $\tau, \tau_1, \tau_2 \in [\Sigma^*]$ *and* $\alpha \in [\Sigma]$ *the following condition holds:*

$\tau\alpha = \tau_1\tau_2$ *iff*

1) $\tau_2 = \tau''\alpha$ *and* $\tau_1\tau'' = \tau$, *for some* $\tau'' \in [\Sigma^*]$ *or*

2) $\tau_1 = \tau'\alpha$, $\{a\} \times \Sigma(\tau_2) \subseteq I$, *and* $\tau'\tau_2 = \tau$, *for some* $\tau' \in [\Sigma^*]$ *and* $[a] = \alpha$.

Proof.(<=) Obvious.
(=>) Let w be any linearisation of $\tau\alpha$. Then w must be of the following form $w = w'aw''$, where $[w'w''] = \tau$, $[a] = \alpha$, and $\{a\} \times \Sigma([w'']) \subseteq I$, (where $\{a\} \times \emptyset = \emptyset$). Now, if we represent w by concatenation of two strings w_1, w_2 s.t. $w = w_1w_2$, then the following cases are possible:

- $w_1 = w_1'$ and $w_2 = w_2'aw''$, where $w' = w_1'w_2'$, or
- $w_1 = w'aw_1''$ and $w_2 = w_2''$, where $w'' = w_1''w_2''$.

Now, let $\tau\alpha = \tau_1\tau_2$. Therefore, either $\tau_1 = [w_1']$ and $\tau_2 = [w_2'aw'']$, where $w' = w_1'w_2'$ and $\{a\} \times \Sigma([w'']) \subseteq I$, or $\tau_1 = [w'aw_1'']$ and $\tau_2 = [w_2'']$, where $w'' = w_1''w_2''$ and $\{a\} \times \Sigma([w'']) \subseteq I$. This implies that either $\tau_1 = [w_1']$ and $\tau_2 = \tau''[a]$, where $\tau'' = [w_2'w'']$, $\tau_1\tau'' = [w_1'w_2'w''] = [w'w''] = \tau$, or $\tau_1 = \tau'[a]$, $\{a\} \times \Sigma([w_2'']) \subseteq I$, and $\tau_2 = [w_2'']$, where $\tau' = [w'w'']$, $\tau'\tau_2 = [w'w_1''w_2''] = [w'w''] = \tau$.

Let $M_G = (F_{w_0}, V_G)$ be a model generator, where $F_{w_0} = (W, \Sigma, \rightarrow, I, w_0)$, and let AC be the acceptance function for F_{w_0}. Let ψ be a formula to be checked over the model, generated by M_G. The general idea of our algorithm consists in defining a new model generator M_G' for which $AC'(\tau) = AC'(\tau')$ implies

- $AC(\downarrow \tau) = AC(\downarrow \tau')$, and
- for each a s.t. $Y_a\varphi \in Subformulas(\psi)$, if $\tau = \tau_1[a]$, then $\tau' = \tau_1'[a]$ and $AC(\tau_1) = AC(\tau_1')$, for some $\tau_1, \tau_1' \in [\Sigma^*]$.

The model generator M_G' is built around the *rcts* accepting $Tr(F_{w_0})$, being the quotient structure of $Tr(F_{w_0})$ by the relation EQ. This relation is defined below.

Let $\tau, \tau' \in Tr(F_{w_0})$ and $\Delta \subseteq \Sigma$. We define:

- $I_\psi(\Sigma) = I(\Sigma) \cup \{\{a\} \subset \Sigma \mid Y_a\varphi \in Subformulas(\psi)\}$,
- a function $^- : 2^\Sigma \leftrightarrow\!\!\!\rightarrow I_\psi(\Sigma)$ s.t. $\overline{\overline{\Delta}} = \Delta$, if $\Delta \in I_\psi(\Sigma)$, $\overline{\Delta} = \Sigma$, if $\Delta \notin I_\psi(\Sigma)$,
- a function $p : [\Sigma^*] \leftrightarrow\!\!\!\rightarrow \{tt, ff\}$ s.t. $p(\tau') = tt$, if $|\tau'| = 1$, $p(\tau') = ff$, if $|\tau'| \neq 1$,
- a function $Rep : [\Sigma^*] \leftrightarrow\!\!\!\rightarrow 2^{W \times I_\psi(\Sigma) \times \{tt,ff\}}$ (tt stands for *true* and ff stands for *false*) s.t.
- $Rep(\tau) = \{(AC(\tau_1), \overline{\Sigma(\tau_2)}, p(\tau_2)) \mid \tau_1\tau_2 = \tau\}$,
- $\tau \; EQ \; \tau'$ iff $Rep(\tau) = Rep(\tau')$,
- $[\tau]_{EQ} = \{\tau' \in Tr(F_{w_0}) \mid \tau \; EQ \; \tau'\}$,
- $states(\tau) = \{w \in W \mid (w, \Delta, q) \in Rep(\tau)$, for any non-empty $\Delta \subseteq \Sigma$ and any $q \in \{tt, ff\}\}$.

  Intuitively speaking, $states(\tau)$ gives the set of all states in W accepting strict prefixes of τ,
- $pred(\tau) = \{(w, \Delta, q) \in Rep(\tau) \mid q = tt\}$,

  Intuitively speaking, $pred(\tau)$ is the set of all states in W accepting predecessors of τ.

The definition of $Rep(\tau)$ gives rise to the equivalence relation satisfying the conditions of Definition 2.1 and carrying information about the past and predecessors of traces. The interested reader can check that $Rep(\tau)$,

defined as $\{AC(\tau') \mid \tau' \leq \tau\}$ or $\{AC(\tau') \mid \tau' \rightarrow \tau\}$ wouldn't give rise to an equivalence relation as required.

Next, it is shown that EQ is indeed an equivalence relation, which can be used for defining a new finite representation of $Tr(F_{w_0})$.

Lemma 7.3 *The following conditions hold:*

i) $\tau\ EQ\ \tau'$ *implies* $AC(\tau) = AC(\tau')$,

ii) EQ *satisfies the conditions of the Definition 2.1,*

iii) $\tau\ EQ\ \tau'$ *implies* $states(\tau) = states(\tau')$ *and* $pred(\tau) = pred(\tau')$.

Proof.

i) Assume that $\tau\ EQ\ \tau'$. From the definition of $Rep(\tau)$, it follows that $AC(\tau) = w$ for $(w, \emptyset, ff) \in Rep(\tau)$ (it is exactly one element in $Rep(\tau)$ of the form $(w, \emptyset, ff)$). As $Rep(\tau) = Rep(\tau')$, it follows that $AC(\tau) = AC(\tau')$.

ii) The index of EQ is obviously finite and moreover, it can be shown to be smaller than $2^{2\times|W|\times|I_\psi(\Sigma)|}$.

Then, we prove that $\tau\ EQ\ \tau'$ and $\tau\alpha \in Tr(F_{w_0})$ implies $\tau\alpha\ EQ\ \tau'\alpha$, for each $\alpha \in [\Sigma]$. To this end we show that $Rep(\tau\alpha)$ can be defined in terms of $Rep(\tau)$, α, and (Σ, I). We need the following notation: for each $w \in W$ and $a \in en(w)$, by $w(a)$ we denote the state $w' \in W$ s.t. $w \xrightarrow{a} w'$.

Now, we come back to the proof of ii). Let $\alpha = [a]$.

- $Rep(\tau\alpha)$ = (by definition) $\{(AC(\tau_1), \overline{\Sigma(\tau_2)}, p(\tau_2)) \mid \tau_1\tau_2 = \tau\alpha\}$ = (by Lemma 7.2)
- $\{(AC(\tau_1), \overline{\Sigma(\tau''\alpha)}, p(\tau''\alpha)) \mid \tau_1\tau'' = \tau\} \cup$
  $\{(AC(\tau'\alpha), \overline{\Sigma(\tau_2)}, p(\tau_2)) \mid \tau'\tau_2 = \tau \text{ and } \{a\}\times\Sigma(\tau_2) \subseteq I\}$
  = (by definition of $Rep(\tau)$)
- $\{(w, \overline{\Delta \cup \{a\}}, emp(\Delta)) \mid (w, \Delta, q) \in Rep(\tau)\} \cup$
  $\{(w(a), \Delta, q) \mid (w, \Delta, q) \in Rep(\tau) \text{ and } \{a\} \times \Delta \subseteq I\}$,
  where $emp(\Delta) = tt$, if $\Delta = \emptyset$, and $emp(\Delta) = ff$, otherwise.

Notice that, $Rep(\tau\alpha) = Rep(\tau'\alpha)$, if $Rep(\tau) = Rep(\tau')$.
iii) It follows directly from the definitions of EQ, $states(\tau)$ and $pred(\tau)$.

Let $F'_{w'_0} = (W', \Sigma, \rightarrow', I, w'_0)$ be the quotient structure of $Tr(F_{w_0})$ by the equivalence relation EQ s.t. elements of W' are of the form $Rep(\tau)$ for $\tau \in Tr(F_{w_0})$ rather than $[\tau]_{EQ}$. We show now that $F'_{w'_0}$ can be defined directly from F_{w_0}.

The following notions simplify the rest of the construction.

We define:

- a function $last : W' \Leftrightarrow\!\!\!\rightarrow W$, s.t. $last(w') = w$, if $(w, \emptyset, ff) \in w'$,
- $REP(w'a) = \{(w, \overline{\Delta \cup \{a\}}, emp(\Delta)) \mid (w, \Delta, q) \in w'\} \cup \{(w(a), \Delta, q) \mid (w, \Delta, q) \in w' \text{ and } \{a\} \times \Delta \subseteq I\}$, for any $w' \in W'$ and $a \in en(last(w'))$,
  (intuitively speaking, $REP(w'a)$ defines the next state in W', reached after executing a at w'),

The construction of $F'_{w'_0}$ is performed inductively, in stages.

Let $F^1_{w'_0} = (W'_1, \Sigma, \emptyset, I, w'_0)$, where $W'_1 = \{w'_0\}$ and $w'_0 = \{(w_0, \emptyset, ff)\}$. Now, for each node w' of $W'_i \Leftrightarrow W'_{i-1}$ (with $W'_0 = \emptyset$) we add all its successors in W' and extend $\rightarrow'_i$, respectively. Let $F^i_{w'_0} = (W'_i, \Sigma, \rightarrow'_i, I, w'_0)$ and $W^{\emptyset}_i = W'_i \Leftrightarrow W'_{i-1}$. Then, $F^{i+1}_{w'_0} = (W'_{i+1}, \Sigma, \rightarrow'_{i+1}, I, w'_0)$, where

- $W'_{i+1} = W'_i \cup \{REP(w'a) \mid w' \in W^{\emptyset}_i,\ a \in en(last(w'))\}$,
- $\rightarrow'_{i+1} = \rightarrow'_i \cup \{(w', a, REP(w'a)) \mid w' \in W^{\emptyset}_i,\ a \in en(last(w'))\}$.

The construction stops at the least m, when $F^m_{w'_0} = F^{m+1}_{w'_0}$. It is easy to see that $F^m_{w'_0}$ is isomorphic to $F'_{w'_0}$.
Then, the new model generator $M'_G = (F'_{w'_0}, V')$ is defined as follows:

- $F'_{w'_0} = (W', \Sigma, \rightarrow', I, w'_0)$,

- $V'_G : W' \Leftrightarrow 2^{PV}$ is a valuation function satisfying the following condition:
  - $p \in V'_G(w')$ iff $p \in V_G(last(w'))$, for $p \in PV$.

7.1 Improving the algorithm

In order to decrease the number of elements of W' we may require that:

- if $(w, \Delta, tt) \in Rep(\tau)$, then $(w, \Delta, ff) \notin Rep(\tau)$,
- if $(w, \Delta, ff) \in Rep(\tau)$, then $(w, \Delta', ff) \notin Rep(\tau)$, for $\Delta \subset \Delta'$ and $\Delta \neq \Delta'$.

The definition of $Rep(\tau)$ with these changes remains correct. To estimate the complexity of the algorithm we have to assess how much it does cost to build M'_G. Therefore, notice that:

- $|W'| \leq 2^{2 \times |W| \times |I_\psi(\Sigma)|}$.
- Checking whether $\{a\} \times \Delta \subseteq I$, for all $a \in \Sigma_I$ and for all $\Delta \in I_\psi(\Sigma)$ costs $|\Sigma_I| \times |I_\psi(\Sigma)| \times |\Sigma_I| \times |I|$.
- Checking whether $\Delta \cup \{a\} \notin I_\psi(\Sigma)$, for all $a \in \Sigma_I$ and for all $\Delta \in I_\psi(\Sigma)$ costs $|\Sigma_I| \times |I_\psi(\Sigma)| \times |\Sigma_I| \times |I|$, since for each $a \in \Sigma_I$ and $\Delta \in I_\psi(\Sigma)$ the algorithm can check whether $(\{a\} \times Con(\Delta)) \cap I \neq \emptyset$, where $Con(\Delta) = \{a' \in \Sigma \mid \{a'\} \times \Delta \subseteq I\}$ ($Con(\Delta)$ has been already computed by the algorithm while checking whether $\{a\} \times \Delta \subseteq I$, for all $a \in \Sigma_I$), which costs at most $|\Sigma_I| \times |I|$.
- At each state w' of W' the cost of calculating the next states is $O(|W| \times |I_\psi(\Sigma)|)$ for each $a \in en(last(w'))$ i.e., $O(| \rightarrow | \times |I_\psi(\Sigma)|)$. Thus, for all states $w' \in W'$ it is $O(| \rightarrow | \times |I_\psi(\Sigma)| \times 2^{2 \times |W| \times |I_\psi(\Sigma)|})$.

Therefore, the complexity of building M'_G is:

$$O(| \rightarrow | \times |I_\psi(\Sigma)| \times 2^{2 \times |W| \times |I_\psi(\Sigma)|}).$$

Notice that for each subformula φ of ψ and for each $\tau, \tau' \in Tr(F_{w_0})$, if $AC'(\tau) = AC'(\tau')$, then $M, \tau \models \varphi$ iff $M, \tau' \models \varphi$. Therefore, we

label $w' \in W'$ with φ (written $M'_G, w' \models \varphi$), if $M, \tau \models \varphi$, for some τ: $AC'(\tau) = w'$.

Obviously, for each subformula φ without past operators, for each $\tau, \tau' \in Tr(F_{w_0})$, if $AC(\tau) = AC(\tau')$, then $M, \tau \models \varphi$ iff $M, \tau' \models \varphi$. Therefore, we label $w \in W$ with φ (written $M_G, w \models \varphi$), if $M, \tau \models \varphi$, for some τ: $AC(\tau) = w$.

We write $M_G \models \varphi$, if $M_G, w \models \varphi$, for all $w \in W$.

Next, we show model checking algorithms. The method is inductive i.e., given a formula ψ, starting from its shortest and most deeply nested subformula φ the algorithm labels these states of M'_G with φ, which accept traces, at which φ holds. Therefore, in case of checking a less nested subformula, it can be assumed that the states have just been labelled with all its subformulas.

Firstly, we label states of M_G with all the subformulas of ψ, which do not contain past subformulas. Then, we label states of M'_G with all the subformulas of ψ. Below, we give algorithms for labelling states of M'_G. These algorithms can be as well applied for labelling states of M_G.

We show how to label states of M'_G with formulas of the form p, $\neg\varphi$, $\varphi \wedge \gamma$, $Y_a\varphi$, $H\varphi$, $EX_a\varphi$, $EG\varphi$, and $E(\varphi U \gamma)$.

7.2 Model checking for p, $\neg\varphi$, $\varphi \wedge \gamma$

There is not a lot to do in the case of checking formulas of the form:

- $p \in PV$,
- $\neg\varphi$,
- $\varphi \wedge \gamma$.

Notice, that:

1. $M'_G, w' \models p$ iff $p \in V'_G(w')$, for $p \in PV$,
2. if φ, γ are state formulas,

   $M'_G, w' \models \neg\varphi$ iff not $M'_G, w' \models \varphi$,

   $M'_G, w' \models \varphi \wedge \gamma$ iff $M'_G, w' \models \varphi$ and $M'_G, w' \models \gamma$.

Therefore, in the first case we check whether p is an element of $V'_G(w')$, in the second case we check whether w' has been labelled with φ and in the third case we check whether w' has been labelled with φ and γ. The complexity of checking a formula of the above form over all states is $O(|W'|)$.

Now, we define algorithms for checking formulas of the form $Y_a\varphi$, $H\varphi$, $EX_a\varphi$, $E(\varphi U\gamma)$, and $EG\varphi$.

7.3 Model checking for $Y_a\varphi$

Observe that $M'_G, w' \models Y_a\varphi$ iff there is $(w, \{a\}, tt) \in w'$ s.t. $M_G, w \models \varphi$. The complexity of labelling states of M'_G with $Y_a\varphi$ is $O(|W'| \times |W| \times |I_\psi(\Sigma)|)$.

7.4 Model checking for $H\varphi$

Observe that $M'_G, w' \models H\varphi$ iff for all $(w, \Delta, q) \in w'$ if $\Delta \neq \emptyset$, then $M_G, w \models \varphi$. The complexity of labelling states of M'_G with $H\varphi$ is $O(|W'| \times |W| \times |I_\psi(\Sigma)|)$.

7.5 Model checking for $EX_a\varphi$

Observe that $M'_G, w' \models EX_a\varphi$ iff there is $w'' \in W'$ s.t. $w' \xrightarrow{a} w''$ and $M'_G, w'' \models \varphi$. Therefore, the algorithm finds all the states at which φ holds and labels all its a-predecessors with $EX_a\varphi$. The complexity of labelling states of M'_G with $EX_a\varphi$ is $O(|W'| + |\rightarrow'|)$.

7.6 Model checking for $E(\varphi U\gamma)$

Observe that $M'_G, w' \models E(\varphi U\gamma)$ iff there is a state $w'' \in W'$ and a sequence of states $w'_0, \ldots, w'_n \in W'$ s.t. $w' = w'_0 \rightarrow \ldots \rightarrow w'_n = w''$ and $M'_G, w'' \models \gamma$, $M'_G, w'_i \models \varphi$ for $0 \leq i < n$.

The above follows from the following lemma:

Lemma 7.4 *Every finite path in $F'_{w'_0}$ can be extended to an observation in $F'_{w'_0}$.*

Proof follows from the fact that every finite path in $Tr(F'_{w'_0})$ can be extended to an observation in $Tr(F'_{w'_0})$ (see [17]).

Firstly, all the states at which γ holds are labelled with $E(\varphi U \gamma)$. Secondly, the algorithm goes backwards using the relation $\rightarrow'^{-1}$ and labels all states at which φ holds with $E(\varphi U \gamma)$. The complexity of labelling states with $E(\varphi U \gamma)$ is $O(|W'| + | \rightarrow' |)$.

7.7 Model checking for $EG\varphi$

We assume that $I \neq \emptyset$. Observe that $M'_G, w' \models EG\varphi$ iff there is an observation x starting at w' s.t. $M'_G, w'' \models \varphi$, for each w'' on x. The model checking algorithm uses Lemma 7.1.

Let $W'_\varphi = \{w' \in W' \mid M'_G, w' \models \varphi\}$ be the subset of W', labelled with φ. Firstly, the subset W'_φ of W' is selected. By a *strongly connected component* in W'_φ we mean any subset $W'' \subseteq W'_\varphi$ satisfying one of the following conditions:

- $(\forall w_1, w_2 \in W'')$: $w_1 \rightarrow'^* w_2$ and $w_2 \rightarrow'^* w_1$, or
- W'' contains only one state w'' s.t. w'' does not have any successor in W'.

Secondly, all the maximal strongly connected components in W'_φ are selected. Notice that they are disjoint. Next, the states of W' are labelled according to the following theorem:

Theorem 7.1 *$M'_G, w'_0 \models EG\varphi$ iff $w'_0 \in W'_\varphi$ and there is a maximal strongly connected component W'' in W'_φ reachable from w'_0 by a path contained in W'_φ and (*): $(\forall a \in \Sigma)(\exists w' \in W'')$ $a \notin en(w')$ or $\{a\} \times en(w', W'') \not\subseteq I$.*

Proof. (=>). If $M'_G, w'_0 \models EG\varphi$, then there is an observation starting at w'_0, say $x = w'_0 a_0 w'_1 a_1 \ldots$, s.t. $(\forall i \geq 0)$ $M'_G, w'_i \models \varphi$. If x is finite, then the last state of x is a maximal strongly connected component in W'_φ satisfying (*).

Thus, let's assume that x is infinite. Let w'' be the first state on x s.t. each $w' \in W'$ either doesn't appear on x after w'' or appears infinitely many times. Then, the set W'' of all states appearing on x after w'' is a strongly connected component. It follows from Lemma 7.1 that W'' satisfies the required property. If W'' is not a maximal strongly connected component, then it can be extended to it and the property in question is preserved.

($<=$). If W'' contains only one state without any successor in W', then obviously $M'_G, w'_0 \models EG\varphi$. So, assume that this is not true. Take any path $x = w'_0 a_0 w'_1 a_1 \ldots$ s.t. $(\forall i \geq 0)\ w'_i \in W'_\varphi$ and $(\forall w', w'' \in W'')$ if $w' \overset{a}{\rightarrow}' w''$, then for infinitly many i: $w'_i = w'$, $w'_{i+1} = w''$, and $a_i = a$. Clearly, by the assumption and by Lemma 7.1, it follows that x is an observation and by construction $x \models G\varphi$.

Selecting W'_φ and maximal strongly connected components costs $O(|W'| + |\rightarrow'|)$ (see [21]). Then, components satisfying the property (*) in the Theorem 7.1 are labelled. This may cost $O((|W'| + |\rightarrow'|) \times |\Sigma| \times |I|)$. If w'_0 is in W'_φ and there is a path from w'_0 to a labelled component in W'_φ, then $M'_G, w'_0 \models EG\varphi$, otherwise $M'_G, w'_0 \models \neg EG\varphi$. Therefore, the complexity of checking $EG\varphi$ is $O(|\Sigma| \times |I| \times (|W'| + |\rightarrow'|))$.

7.7.1 Improving the algorithm

Now, it is shown how to improve the algorithm. The above algorithm will be more efficient, if instead of checking for all $a \in \Sigma$, it will check only for those a, for which it exists at least one independent action i.e., $a \in \Sigma_I$. If $a \notin \Sigma_I$, then $\{a\} \times en(w', W'') \not\subseteq I$, so this does not need to be checked. Finding the set Σ_I costs $O(|I|)$. Therefore, the complexity of the improved algorithm is

$$O(|\Sigma_I| \times |I| \times (|W'| + |\rightarrow'|)).$$

It was assumed that $I \neq \emptyset$. In the other case, each path is an observation, so to prove that $EG\varphi$ holds does not require to check any property

about maximal strongly connected components. Then, the complexity is $O(|W'| + |\rightarrow'|)$.

7.8 Complexity of CTL$_{P-}$ model checking

In order to handle an arbitrary CTL$_{P-}$ formula ψ, the state-labelling algorithm is applied to the subformulas of ψ starting with the shortest and most deeply nested one. Since $|\Sigma_I| \times |I| \leq |I_\psi(\Sigma)|$ and $|\rightarrow'| \leq |\rightarrow| \times |W'|$, then each pass does not take more time than $O(|\rightarrow| \times |I_\psi(\Sigma)| \times 2^{2\times|W|\times|I_\psi(\Sigma)|})$, and since ψ contains at most $lenght(\psi)$ different subformulas, the algorithm requires time

$$O(lenght(\psi) \times |\rightarrow| \times |I_\psi(\Sigma)| \times 2^{2\times|W|\times|I_\psi(\Sigma)|}).$$

Example 7.1 (proving serializability) *Below, it is shown how to prove serializability of the trace system T_2.*

We have to check whether:

1. $M_2, [\epsilon] \models p_1 \wedge p_2 \wedge p_9 \Rightarrow AFP((p_7 \wedge p_2 \wedge p_9) \vee (p_1 \wedge p_8 \wedge p_9))$,
2. $M_2, [\epsilon] \models AG[((p_7 \wedge p_2 \wedge p_9) \vee (p_1 \wedge p_8 \wedge p_9)) \Rightarrow AF(P(p_7 \wedge p_8 \wedge p_9) \vee (p_7 \wedge p_8 \wedge p_9))]$.

Fortunately, in this case M_2 is equall to its model generator. Therefore, we can carry out all the proofs over M_2. The only difficult thing is to label states of M_2 with the formulas:

1. $AFP((p_7 \wedge p_2 \wedge p_9) \vee (p_1 \wedge p_8 \wedge p_9))$,
2. $AF(P(p_7 \wedge p_8 \wedge p_9) \vee (p_7 \wedge p_8 \wedge p_9))$.

It is shown how to do that for the formula 1. First notice that: $AFP((p_7 \wedge p_2 \wedge p_9) \vee (p_1 \wedge p_8 \wedge p_9)) \equiv \neg EGH\varphi$, where $\varphi = (\neg p_7 \vee \neg p_2 \vee \neg p_9) \wedge (\neg p_1 \vee \neg p_8 \vee \neg p_9)$. Next, states of M_2 are labelled with $EGH\varphi$. Notice that $\varphi \in V_2(\tau)$, if $\tau \notin \{[a_1a_2a_3], [b_1b_2b_3]\}$. Therefore, $M_2, \tau \models H\varphi$ iff $\tau \notin (\uparrow [a_1a_2a_3] \cup \uparrow [b_1b_2b_3]) \Leftrightarrow ([a_1a_2a_3] \cup [b_1b_2b_3])$. Now, using the algorithm for $EG\varphi$ in M_2, we find that $M_2, [\epsilon] \not\models EGH\varphi$. Thus, $M_2, [\epsilon] \models AFP((p_7 \wedge p_2 \wedge p_9) \vee (p_1 \wedge p_8 \wedge p_9))$. Consequently, we get $M_2, [\epsilon] \models p_1 \wedge p_2 \wedge p_9 \Rightarrow AFP((p_7 \wedge p_2 \wedge p_9) \vee (p_1 \wedge p_8 \wedge p_9))$.

7.9 Extending model checking to CTL_P and CTL^*_P

Our method of model checking can be easily extended s.t.:

1. past formulas can be nested - (model checking for CTL_P),
2. future formulas are those of CTL* [47] (CTL with nested path formulas) -

   (model checking for $\mathrm{CTL}^*{}_{P-}$),
3. all the above extensions together - (model checking for $\mathrm{CTL}^*{}_P$).

For 1), we define a sequence of unfoldings $M_1, \ldots, M_n$, where $M_1 = M'_G$, n is the maximal depth of nested past formulas in ψ, M_{i+1} is obtained from M_i in the same way as M'_G was obtained from M_G. Then, we inductively label states of M_i with subformulas of ψ containing nested past formulas of depth at most i. Then, $M, [\epsilon] \models \psi$ iff the beginning state of M_n is labelled with ψ. In the worst case we can arrive at the complexity $exp_n(2 \times |W| \times |I_\psi(\Sigma)|)$.

In this case, the meaning of our result is only theoretical, but, at least, we have shown that it is decidable whether any CTL_P formula is true in a finite state trace model. This result seems to be interesting on its own, especially in comparison with the result of chapter 9.

For 2), after unwinding the model generator, we treat past formulas as fresh propositions and apply the standard methods for CTL* model checking [24]. Then, we arrive at the complexity exponential in the number of states and exponential in the number of subformulas of ψ.

For 3), we combine the methods of 1) and 2).

8 Undecidability of CTL_P

We now turn to the problem of determining the satisfiability of CTL_P formulas. This problem may be stated as: Given a CTL_P formula φ, is there a model $M = (F_{v_0}, V)$ s.t. $M, v_0 \models \varphi$. If φ is true at v_0 of M, M is said to be a model of φ. Note also that the CTL_P formula φ is satisfiable iff $\neg\varphi$ is not valid, hence exhibiting a decision procedure for satisfiability amounts to deciding the validity problem.

Unfortunately, to our big surprise we have to report a negative result. CTL_P is not decidable ! It turns out that we can encode a grid using our language.

Consider a Petri Net N, composed of two independent transitions a and b, each of them can be executed infinitely many times. The trace system $\mathcal{T}$ giving the semantics to N is composed of all the traces belonging to the language $[\Delta^*]$, where $\Delta = \{a, b\}$ and $I = \Delta \times \Delta \Leftrightarrow id_\Delta$. A frame for $\mathcal{T}$ is defined as $F_{[\epsilon]} = ([\Delta^*], \rightarrow, [\epsilon])$, It is possible to characterize this frame up to isomorphism. But, then one can encode the following *recurring tiling problem*, which has been shown to be undecidable in [43]. Below, we follow the definition of [44] and [45].

Let , be a finite set of types of tiles such that for each $T \in$, each side - North, East, South and West - is assigned a number (N(T), E(T), S(T), and W(T), resp.). Let $Co \subseteq$, ^{4} be a set of colors such that $c = (T_1, T_2, T_3, T_4) \in Co$ iff $S(T_1) = N(T_3)$, $E(T_1) = W(T_2)$, $N(T_4) = S(T_2)$, and $W(T_4) = E(T_3)$ (see the picture below).

T_1	T_2
T_3	T_4

For each $c = (T_1, T_2, T_3, T_4) \in Co$ let $U(c), R(c) \subseteq Co$, where U stands for Up and R stands for $Right$, with $U(c) = \{c' \in Co \mid c' = (T, T', T_1, T_2)$, for some $T, T'\}$, $R(c) = \{c' \in Co \mid c' = (T_2, T, T_4, T')$, for some $T, T'\}$. Let $T_0, T_f \in$, be two special types. The problem is to find a coloring $c : I\!N \times I\!N \Leftrightarrow\!\!\!\rightarrow Co$ such that for all $i, j \in N$, $c(i, j) \in Co$, $c(0, 0) \in \{c' \in Co \mid c' = (T, T', T_0, T'')$, for some $T, T', T''\}$, $c(i, j+1) \in U(c(i, j))$, and $c(i+1, j) \in R(c(i, j))$, and there are infinitely

many colors in the leftmost column, which contain the tile type T_f. The above formulation means that one has to exhibit a coloring of the lattice points in the plane such that if a point has a color c, then the point just above has a color from the set $U(c)$ and the point to the right has a color from the set $R(c)$, the beginning has a color of a given subset of Co and the given tile type T_f occurs infinitely often in the leftmost column.

Let $PV = \{C_i \mid c_i \in Co\}$. Now, it is possible to give a set of formulas of CTL_{P-} s.t. its conjunction is satisfiable iff the recurring tiling problem has a solution. Firstly, we give formulas encoding the grid:

(A):

1. $AG(EX_a true \wedge EX_b true)$
2. $AG(\bigwedge_{c \in \Sigma - \{a,b\}} \neg EX_c true)$,
3. $EF(Y_a true \wedge Y_b true)$

1) expresses that two actions a and b are executed at each state. Since our model is a partially ordered set, therefore it contains infinitely many states. It follows from 2) that each of them has exactly two successors. 3) specifies that a and b are independent at some state. Thus, by definition of trace systems a and b are independent at each state. So, the frame is a grid representing a trace system $[\{a, b\}^*]$.

Now, we give formulas describing the tiling:

(B):

1. $\bigvee\{C_i \mid c_i = (T, T', T_0, T'')$, for some $T, T', T''\}$,
2. $AG(\bigvee(\{C_i \mid c_i \in Co\}) \wedge (\bigwedge\{C_i \Rightarrow \neg C_j \mid i \neq j\}))$,
3. $AG(\bigwedge(\{C_i \Rightarrow EX_a(\bigvee\{C_j \mid c_j \in U(c_i)\}) \mid c_i \in Co\}))$,
4. $AG(\bigwedge(\{C_i \Rightarrow EX_b(\bigvee\{C_j \mid c_j \in R(c_i)\}) \mid c_i \in Co\}))$,
5. $AG(\neg Y_b true \Rightarrow EF(\neg Y_b true \wedge \bigvee\{C_i \mid$ type T_f occurs in color $c_i\}))$.

1) expresses that the tile type T at the beginning is T_0. 2) enforces exactly one color at each point of the grid. 3) and 4) ensures that

successors have the right color. 5) requires that the tile type T_f occurs infinitely often in the leftmost column.

The recurring tiling problem has a solution iff the conjunction of our formulas A) and B) is satisfiable. As in [44], it follows that the validity problem for CTL_{P-} is $\pi_1^1 \Leftrightarrow hard$.

It can be easily shown that our logic remains undecidable, even if it does not contain backward step operators, next step operators are not labelled, but a valuation function encodes labelling as shown in Example 4.1. Then, the formulas 1), 2) in A) and 3), 4) in B) can use unlabelled versions of next step operators (see Example 4.1), the formulas 3) in A) and 5) in B) can be replaced by:

3'. $EXEX(p_a \wedge p_b \wedge P(p_a \wedge \neg p_b) \wedge P(p_b \wedge \neg p_a))$,

5'. $AG(H(\neg p_b) \wedge \neg p_b \Rightarrow EF(H(\neg p_b) \wedge \neg p_b \wedge \bigvee\{C_i \mid \text{type } T_f \text{ occurs in color } c_i\}))$.

The undecidability result explains why we couldn't use any standard methods for model checking.

9 Comparing CTL_P with other logics

CTL_P contains CTL and, as it was said before, it can be seen as a restricted version of the logic, defined in [16]. The language of CTL_P is an extension of that of Hennessy-Milner logic with backward modalities and a restriction of the language of POTL [3, 4]. CTL_P resembles also the partial order logic (P.O.-logic), introduced by A. Sinachopoulos [14], which, however, does not contain path quantifiers. A similar logic, but with run operators (PN-logic), has been also defined by Reisig in [11,12]. CTL_P differs from ISTL [13] in the definition of a frame; ISTL is interpreted on runs of trace systems.

Our results on undecidability of checking satisfiability as well as the method of model checking can be easily extended on propositional versions of the following logics: Hennessy-Milner logic with backward modalities [32], ISTL [13], P.O.-logic [14], and the logic defined in [16], interpreted over trace systems.

10 Final remarks

The presented approach to model checking for a partial order logic with past modalities over structures of global states is the first one, known from the literature. In [4] model checking for a similar logic, interpreted over local state models has been investigated. Our paper is also the first one showing how to prove properties of partial order executions, of serializability, and snapshots by model checking. Therefore, it is not possible to compare the complexity of the model checking algorithm with others.

Our method can be viewed as an extension and refinement of the method of Clarke and al. [21] to cover also partial order properties. The alternative approach would be to apply automata-theoretic techniques in the style of [20] or [31]. This, however, cannot be done immediately as it is not clear how to build automata accepting CTL_P formulas.

We have proved that model checking becomes NP-hard as soon as we have introduced backward modalities. Moreover, unfortunately, the best algorithm we could give is of exponential complexity in the number of states of a model and linear in the lenght of a formula. Therefore, our algorithm can be applied for systems with not too many states. It seems to be impossible to define an algorithm linear in the number of states of a model (and, obviously, exponential in the lenght of a formula).

Our undecidability result shows that in general it is not possible to synthesise systems from their specifications in CTL_{P-}, or any similar logic, e.g., defined in [13], [16]. Therefore, altough, serializability and run properties can be proved by model checking, they cannot be imposed on systems, synthesised from their specifications.

ACKNOWLEDGEMENTS: The author wishes to thank dr. Ruurd Kuiper for improving the language of the paper. Special thanks are directed to my wife Agnieszka for the help in designing the figures.

11 References

[1]: Manna, Z., Pnueli, A., The Anchored Version of the Temporal

Framework, LNCS 354, 1988.

[2]: Emerson, E.A., Srinivasan, J., Branching Time Temporal Logic, LNCS 354, 1988.

[3]: Pinter, S.S., Wolper,P., A Temporal Logic for Reasoning about Partially Ordered Computations, Proc. 3rd Symp. on Principles of Distributed Computing, pp. 28-37, Vancouver 1984.

[4]: Kornatzky, Y., Pinter, S.S., A Model Checker for Partial Order Temporal Logic, EE Pub n. 597, Department of Electrical Engineering, Technion - Israel Institute of Technology, 1986.

[5]: Katz, S., Peled, D., Interleaving Set Temporal Logic, 6th ACM Symposium on Principles of Distributed Computing, Vancouver Canada, pp. 178-190, 1987.

[6]: Katz, S., Peled, D., An Efficient Verification Method for Parallel and Distributed Programs, LNCS 354, 1988.

[7]: Lodaya, K., Thiagarajan, P.S., A Modal Logic for a Subclass of Event Structures, LNCS 267, pp. 290-303, 1987.

[8]: Penczek, W., A Temporal Logic for The Local Specification of Concurrent Systems, Information Processing, pp. 857-862, IFIP, 1989.

[9]: Penczek, W., A Concurrent Branching Time Temporal Logic, Proceedings of the Workshop on Computer Science Logic, Kaiserslautern, LNCS 440, pp. 337-354, 1990.

[10]: Mukund, M., Thiagarajan, P.S., An Axiomatization of Event Structures, LNCS 405, 1989.

[11]: Reisig, W., Temporal Logic and Causality in Concurrent Systems, LNCS 335, 1988.

[12]: Reisig, W., Towards a Temporal Logic of Causality and Choice in Distributed Systems, LNCS 354, pp. 606-627, 1989.

[13]: Peled, D., Pnueli, A., Proving Partial Order Liveness Properties, Proc. of ICALP, pp. 553-571, 1990.

[14]: Sinachopoulos A, Partial Order Logics for Elementary Net Systems: State- and Event - approches, Proc. of CONCUR'90, 1990.

[15]: Penczek, W., Proving Partial Order Properties Using CCTL, manuscript, 1991.

[16]: Peled, D., Katz, S., and Pnueli, A., Specifying and Proving Serializability in Temporal Logic, Proc. of LICS, 1991.

[17]: Mazurkiewicz, A., Basic Notions of Trace Theory, LNCS 354, pp. 285-363, 1988.

[18]: Mazurkiewicz, A., Ochmanski, E., Penczek, W., Concurrent Systems and Inevitability, TCS 64, pp. 281-304, 1989.

[19]: Lichtenstein, O., and Pnueli, A., Checking that Finite State Concurrent Programs Satisfy their Linear Specification. Proc. of the 12th ACM Symposium on Principles of Programming Languages, pp. 97-107, New Orleans, 1985.

[20]: Vardi, M.Y., Wolper. P., An Automata-Theoretic Approach to Automatic Program Verification, Proc. of LICS, pp. 322-331, 1986.

[21]: Clarke, E.M., Emerson, E.A., Sistla, A.P., Automatic Verification of Finite State Concurrent Systems Using Temporal Logic Specifications: A Practical Approach, Proc. 10th Annual ACM Symp. on Principles of Programming Languages, Austin, pp. 117-126, 1983, and ACM Transactions on Programming Languages and Systems, 8(2), pp. 244-263, 1986.

[22]: Sistla, A.P., Clarke, E., The Complexity of Propositional Temporal Logic, 14th ACM Symposium on Theory of Computing, May 1982, pp. 159-167.

[23]: Emerson, E.A., Clarke, E.M., Using Branching Time Logic to Synthesize Synchronization Skeletons, Science of Computer Programming, vol. 2, pp. 241-266, 1982.

[24]: Emerson, E.A., Lei, C.L., Modalities for Model Checking: Branching Time Logics Strikes Back, Science of Comp. Programming, vol. 8, pp. 275 - 306, 1987.

[25]: Emerson, E.A., Lei, C.L., Efficient Model Checking in Fragments of the Propositional μ-calculus, Proc. of LICS, 1986.

[26]: Emerson, E.A., Lei, C.L., Temporal Reasoning Under Generalized Fairness Constraints, LNCS 210, pp. 21-36, 1986.

[27]: Stirling, C., Walker, A General Tableau Technique for Verifying Temporal Properties of Concurrent Programs, in Semantics for Concurrency, eds. M.Z. Kwiatkowska, M.W. Shields, and R.M. Thomas, pp. 1-15, Leicester 1990.

[28]: Burch, J.R., Clarke E.M., McMillan, K.L., Dill, D.L., and Hwang L.J., Symbolic Model Checking: 10^{20} States and Beyond. Proc. of LICS, 1990.

[29]: Clarke, E.M., and Grumberg, O., Avoiding the State Explosion Problem in Temporal Logic Model-Checking Algorithms. Proc of 5th ACM Symposium on Principles of Distributed Computing, pp. 293-303, 1987.

[30]: Valmari, A., A Stubborn Attack on State Explosion, Proc. of Workshop on Computer Aided Verification, Rutgers, 1990.

[31]: Godefroid, P., Wolper, P., A Partial Approach to Model Checking, Proc. of LICS, 1991.

[32]: Hennessy, M., and Stirling, C., The Power of the Future Perfect in Program Logics, Information and Control 67, pp. 23-52, 1985.

[33]: de Nicola, R., Vaandrager, F., Three Logics for Branching Bisimulation, Proc. of LICS, 1990.

[34]: de Nicola, R., Montanari, U., and Vaandrager, F., Back and Forth Bisimulations, Proc. of CONCUR'90, 1990.

[35]: Emerson, E.A., Halpern, J.Y., Decision Procedures and Expressiveness in the Temporal Logic of Branching Time, Journal of Computer and System Sciences 30, pp. 1-24, 1985.

[36]: Courcoubetis, C., Vardi, M.Y., and Wolper, P., Reasoning about Fair Concurrent Programs, Proc. of the 18th Annual ACM Symp. on Theory on Computing, pp. 283-294, 1986.

[37]: Godefroid, P., Using Partial Orders to Improve Automatic Verification Methods, Proc. of Computer-Aided Verification Workshop, Rutgers, New Jersey, 1990.

[38]: Wolper, P., On the Relation of Programs and Computations to Models of Temporal Logic, LNCS 398, pp. 75 -123, 1987.

[39]: Kwiatkowska, M., Fairness for Non-Interleaving Concurrency, PhD Thesis, University of Leicester, 1989.

[40]: Bednarczyk, M., Categories of Asynchronous Transition Systems, Ph.d. thesis, University of Sussex, 1987.

[41]: Droste M., Concurrency, Automata and Domains, LNCS 443, pp. 195 - 208, 1990.

[42]: Stark, E.W., Concurrent Transition Systems, TCS 64, pp. 221-269, 1989.

[43]: Harel, D., Recurring Dominoes: Making the Highly Undecidable Highly Understandable, Annals of discrete mathematics, 24, pp. 51 - 72, 1985.

[44]: Parikh, R., Decidability and Undecidability in Distributed Transition Systems, A perspective in theoretical computer science, Vol. 16, 1988.

[45]: Paech, B., Concurrency as a Modality, Ph. d. thesis, Munchen University, 1991.

[46]: Aalbersberg I.J., Rozenberg G., Theory of traces, TCS 60, pp. 1-82, 1988.

[47]: Emerson, E.A., Halpern, J.Y., Sometimes and "Not Never" Revisited: On Branching versus Linear Time Temporal Logic, Journal of the ACM 33 (1), pp. 151-178, 1986.

Two extensions of the existing trace model

(Abstract)

Volker Diekert

Institut für Informatik
Universität Stuttgart
Breitwiesenstr. 20 - 22
D-7000 Stuttgart 80

April 1, 1992

The aim of this abstract is to sketch some ideas how to extend the existing trace model in two different directions. Although not done here, we would like to emphasize that both extensions could be combined from a quite general viewpoint. We will give neither proofs nor other details. This will be done in a forthcomming paper. For references we use standard references on traces, semi-traces and partial order sematics without any explicit citation. The related work can be found elsewhere.

1 Philosophy

One of the drawbacks of the existing trace model is that we are not able to express dynamic concurrency. A possible solution is to use pomsets (= partial words = labelled partial orders) for a description of a concurrent process. This enlarges of course the expressive power, but we pay this by loosing many nice algebraic and combinatorial properties of traces. so we risk to loose what make this model attractive and feasable. The original idea of Mazurkiewicz is collecting different sequential observations (or runs) into a single trace. However, this can be applied to pomsets, too.

At first glance this seems to be even a more complicated model than pomsets; but since the model becomes more abstract, we obtain in fact a simplification.

2 Formality

Our starting point is a fixed (finite) semi-dependence alphabet (Σ, D). This corresponds to the static independence relation which is known, say from the given net topology of a concurrent system. For example, if a concurrent system is specified by a Petri net (place/transition net) then we obtain a semi-dependency between transitions a and b soon as one output place of a is an input place of b. On the other hand, if $a \neq b$ and there is no such place, then we know that any sequential observation $uabv$ would give rise to another possible execution $ubav$, or more precisely, the execution of a and b has been concurrently. (N.b. that concurrency does not mean *at the same time*.)

The difference between a semi-dependence relation and a dependence (without the prefix *semi-*) relation is simply that we do not require symmetry of D. However, we will keep the usual technical restriction that D is a reflexive relation. Given (Σ, D), the set of semi-traces is the set of finite acyclic labelled graphs $[V, E, \lambda]$ where $V = \{1, \ldots, n\}$ is the set of vertices, $\lambda : V \to E$ is the labelling and edges are from i to j if and only if $i < j$ and $(\lambda(i), \lambda(j)) \in D$.

Since the static dependency does not reflect the full concurrent behaviour, it may happen that in the modelized process i and j were independent, although $i < j$ and $(\lambda(i), \lambda(j)) \in D$. Thus, we will consider the following. The objects are labelled acyclic graphs $[V, E, \lambda]$ as above; but we require ony that an edge from i to j implies $(\lambda(i), \lambda(j)) \in D$ and that edges are always between vertices with the same label. We call such a graph a *partial trace* over (Σ, D). Moreover, we identify such a graph with its induced partial order. Therefore, partial traces form a subset of those pom-sets over Σ where identically labelled vertices are totally ordered (= semi-words over Σ). However, instead of thinking of subsets, the much better viewpoint is to think of a partial trace as an equivalence class of semi-words. This equivalence is obtained by a forget operator

which simply forgets the ordering between actions a and b if $(a,b) \notin D$.

On the set of partial traces we have at least three useful operations. The most basic one is the concatenation or maybe better: composition. We define $[V_1, E_1, \lambda_1][V_2, E_2, \lambda_2]$ by the disjoint union of labelled graphs with new edges from $i \in V_1$ to $j \in V_2$ whenever $(\lambda(i), \lambda(j)) \in D$. This defines a monoid of partial traces $P(\Sigma, D)$ where the neutral element is the empty graph $[\emptyset, \emptyset, \emptyset]$.

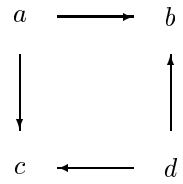
The second important operation is derivation: Given a partial trace $[V, E, \lambda]$ we are allowed to introduce a new edge from i to j if $(\lambda(i), \lambda(j)) \in D$ and if in addition the resulting graph rests acyclic. This corresponds to a linearization operator or to a weakening of the concurrency.

The third operation is synchronization. Here we use essentially the assumption that vertices with the same label are totally ordered. We can synchronize partial traces $[V_1, E_1, \lambda_1]$ and $[V_2, E_2, \lambda_2]$only if for all $a \in \lambda_1(V_1) \cap \lambda_2(V_2)$ the number of vertices with label a is equal. Then we can identify these vertices according to their ordering and we demand that the union of the graphs rests acyclic.

In order to see that the concept above is useful, assume that (Σ, D) is specified through a P/T-system. Then one can define whether a pomset over Σ is enabled and in this case its execution is defined. The basic property, which becomes crucial here, is that one can speak of an enabled partial trace since one can show that no or all representing pomsets are enabled. It is also easy to see that the set of enabled partial traces is closed with respect to derivation. Furthermore, the synchronization operator for nets corresponds to the synchronization of partial traces.

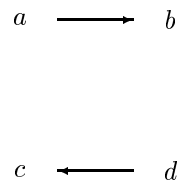
In fact, synchronization can be viewed as another motivation to generalize the semi-trace model. Indeed, contrary to traces, semi-traces are not closed with respect to synchronization. The synchronization of two semi-traces may yield a partial trace which is no semi-trace anymore. This can be seen from the following example.

(Σ, D) =



The synchronization of the two semi-traces $s_1 = [cabd\}$ and $s_2 = [bdca\}$ yields the partial trace

$s_1 \parallel s_2$ =



The reader may verify that this is no semi-trace anymore. It is also a simple exercise to give a Petri net interpretation for this phenomenon.

3 Functoriality

In the following let us analyse some algebraic properties of partial traces. The key observation is that partial traces have a length and that one can prove Levi's lemma. Hence by a well-known result of Christine Duboc partial traces form a free partially commutative monoid. Thus, although considering much more general objects than traces, we stay still inside the theory of free partially commutative monoids. There is only one additional difficulty. We have to consider infinitely generated monoids, in general.

Using functorial properties this can be avoided to some extent. There is an embedding of the monoid of partial traces into a finite directed product of free monoids which are generated by at most two letters,

only. This can be explained as follows. First, we define a morphism between semi-dependence alphabets $h : (\Sigma', D') \leftrightarrows\!\!\!\rightarrow (\Sigma, D)$ to be a mapping $h : \Sigma' \to \Sigma$ such that $(a, b) \in D'$ implies $(h(a), h(b)) \in D$. This yields a contra-variant functor, since h defines a monoid homomorphism $h^* : P(\Sigma, D) \leftrightarrows\!\!\!\rightarrow P(\Sigma', D')$ in the following way.

Given $[V, E, \lambda] \in P(\Sigma, D)$ the partial trace $h^*([V, E, \lambda])$ is obtained by replacing each vertex a by the set $h^{-1}(a)$ without any edge, (thus by a step). For $a' \in h^{-1}(a)$ and $b' \in h^{-1}(b)$ an edge is introduced if and only if $(a', b') \in D'$ and there has been an edge from the corresponding a to b in $[V, E, \lambda]$. The interesting fact (and main theorem on this construction) is that h^* is injective if and only if h is surjective on vertices and edges.

Now, let (Σ, D) be any semi-dependence alphabet. Then (Σ, D) can be covered by the disjoint union of directed edges and some isolated points. This covering can be expressed by a surjective morphism between semi-dependence alphabets. The functorial game above then yields an embedding of $P(\Sigma, D)$ into a finite directed product of monoids which are either of type $P(a \to b)$ (i.e. of type $P(\{a, b\}, \{(a, b)\})$) or isomorphic to a^*. Thus, it is enough to have a closer look at $P(a \to b)$ for two letters a, b.

It turns out that $P(a \to b)$ is isomorphic to $\{a, b\}^*$, the free monoid on two letters. The derivation becomes a semi-Thue system. Again we leave this as an exercise as well as to show that $P(a \leftrightarrow b)$, where the dependence relation is symmetric, is an infinitely generated free monoid. Thus, $P(a \leftrightarrow b)$ is definitely not isomorphic to $\{a, b\}^*$

This functorial approach can be viewed as a synchronization and we obtain that any partial trace is the synchronization of a tuple of words. The translation to P/T systems yields that a process which is described by a partial trace is exactly the synchronization of certain sequences which are recorded locally at places.

4 Compositionality

For the rest of this abstract we briefly indicate a possible extension of the existing trace model to capture some aspects of non-terminating processes.

We restrict ourselves to the classical case of traces over a symmetric dependence relation $D \subseteq \Sigma \times \Sigma$. This restriction is in fact not necessary and a more general theory over partial traces would be possible. However, since anyhow we sketch ideas only, full generality does not seem to be appropriate here.

We consider infinite (real) traces. They are approximated by their finite prefixes, or what is the same, we consider infinite directed sets (with respect to prefix ordering) of finite traces. It is well-known that a meaningfull concatenation is not possible for such objects and this led to the notion of *complex trace*. A ($\alpha\Leftrightarrow$) complex trace is roughly a pair (r, A) where r is a real trace (i.e. a finite or infinite trace) and A is a subset of Σ. Not every set A is allowed, basically we will demand that A contains all letters which appear infinitely often in r.

The semantical idea is that the (finite) set A represents the smallest set of actions we always have to wait for before we can terminate the process r. Thus, for r infinite we have $A \neq \emptyset$ and of course we can never terminate. However, what is possible is that if action b is independent of A then we can perform b in parallel to r after some finite time. This leads to the following calculus

$$(r, A)(s, B) = (r \cdot \mu_A(s), A \cup B \cup \sigma_A(s))$$

where $\mu_A(s)$ is the maximal prefix of s which is independent of A and $\sigma_A(s)$ is the alphabet of the suffix $\mu_A(s)^{-1}s$.

Using prefix ordering on complex traces we obtain a Scott domain. However, as it is known from words, the concatenation is not continuous. This is not surprising, since if $r' \leq r$ and $s' \leq s$ then we can not expect that $r's'$ is a prefix of rs. So, in some sense it gives no information about the process rs we are interested in. The idea to solve this problem can be explained with the help of complex traces. Let us assume that we describe a process by a complex trace (r, A). What we will do is to approximate (r, A) by a sequence $(r_i, A_i)_{i\geq 1}$ where $r_i \leq r$ is a finite prefix and A_i is the alphabetic information about the future, is $A_i = \text{alphabet}(r_{-1}r) \cap A$. Now, it is not realistic to assume that the exact information about A_i is available all the time (or at any time).

The question is whether we should be content with a larger or smaller

set than the actual A_i. A first guess might be *smaller*. We claim that this is false for the application we have in mind. Consider simply the extreme case: without knowing explicitly that a process has terminated we should wait. Something might still happen and we should not start a second process, if it depends on the result of the first one. Thus, the solution larger is more safe. This leads to the following ordering: $(r, A) \leq (s, B)$ if and only if $r \leq s$ and $(B \cup \text{alphabet}(r^{-1}s)) \subseteq A$.

This ordering looks asymmetric, but in the interpretation of the second component as *wait for these possible actions*, it is clear that if r is a finite process and $A \subseteq B$, then (r, A) is a better approximation of r than (r, B). The pair $(r, \emptyset)$ is the exact information about r including the explicit information about termination.

The mathematical counterpart to this somewhat philosophical reasons to study such objects is that we obtain a complex-like domain where concatenation is continuous. In particular, if $(r', A') \leq (r, A), (s', B') \leq (s, B)$ in this new ordering, then the complex product $(r'\mu_{A'}, (s'), A' \cup B' \cup \sigma_{A'}(s'))$ is a better approximation of the composed process $(r, A)(s, B)$ than (r, A), in general.

5 Reality

Whether or not the ideas sketched above will rest a mathematical game or will lead to a useful formalism with realistic applications is open for the moment. We hopefully think that this question will have a positive answer and that this will become more clear, once this abstract will have been transformed into a complete paper.

Rational and Recognizable Complex Trace Languages*

Volker Diekert
Universität Stuttgart
Institut für Informatik
Breitwiesenstr. 20-22
D-7000 Stuttgart 80

Paul Gastin
Université Paris 6
LITP, Institut Blaise Pascal
4, place Jussieu
F-75252 Paris Cedex 05

Antoine Petit
Université Paris Sud
LRI, URA CNRS 410
Bât. 490
F-91405 Orsay Cedex

April 1992

Abstract

Mazurkiewicz defined traces as an algebraic model of finite concurrent processes. In order to modelize non-terminating processes a good notion of infinite trace was needed, which finally led to the notion of complex trace. For complex traces an associative concatenation and an ω-iteration are defined.

This paper defines and investigates rational and recognizable complex trace languages. We prove various closure results such as the closure under boolean operations (for recognizable languages),

*This research has been supported by the ESPRIT Basic Research Actions No. 3166 ASMICS and No. 3148 DEMON.

concatenation, and left and right quotients by recognizable sets. Then we study sufficient conditions ensuring the recognizability of the finite and infinite iterations of complex trace languages. We introduce a generalization of the notion of concurrent iteration which leads to the main result of the paper: the generalization of Kleene's and Ochmanski's theorems to complex trace languages.

0 Introduction

The concept of traces has been introduced by A. Mazurkiewicz [Maz77] as a suitable semantics for non-sequential processes. Let us refer, for instance, surveys [Maz87, AR88, Per89] or the monograph [Die90]. Also, in these references, an extensive bibliography on the subject is given.

There are at least two possible ways to see a trace. A trace can be considered as the set of all possible sequential observations of a concurrent process. The semantics of each sequential observation is given, in a classical way, by a finite word. Hence, a trace is an equivalence class of words. From a different viewpoint, we may see a trace as a finite labeled acyclic graph where edges represent the (causal) dependency of actions. Such a graph is called a dependence graph and the semantics is that an execution has to respect the induced partial order. In this way, the process is modeled as a labeled partial order with an explicit description of concurrency.

Whatever the approach chosen, there is a natural definition of the concatenation of two traces and the set of traces forms a monoid. In fact, this monoid has been introduced and studied independently by Cartier and Foata in combinatorics [CF69].

Recognizable languages describe the behavior of finite state systems and hence form one of the basic families of a monoid. For trace monoids, this family is closed under boolean operations and concatenation, [Fli74, CP85], but it turns out that a trace language T may be recognizable whereas T^* is not. The family of recognizable languages is in fact strictly included in the family of rational trace languages which is the smallest family of trace languages containing the finite languages and which is closed under union, concatenation, and star-iteration. In order to gen-

eralize Kleene's theorem to trace languages it was then necessary to find a new operation instead of star-iteration. In a first step, Métivier and Ochmanski, [Mét86, Och85], proved independently that T^* is recognizable if T is recognizable and consists of connected traces, only. This led Ochmanski to introduce a concurrent version of the star-iteration, called the c-star in the following. The c-star of T is the usual star taken over all connected components of traces in T. Replacing simply the star by the c-star operation, Ochmanski obtains the equality of recognizable and c-rational trace languages [Och85]. Let us mention that the search for sufficient conditions ensuring the recognizability of T^* is an ongoing work, [Sak87, Och90, MR91].

In order to describe non-sequential processes which never terminate, e.g. distributed operating systems, the notion of infinite trace was needed. It seems that the first explicit definition of infinite traces was given by Mazurkiewicz [Maz87]. He defines an (infinite) trace as an (infinite) prefix closed directed subset of finite traces. This is the same as an (infinite) dependence graph where each vertex has finitely many predecessors only, [Maz87, Thm.13]. In the following these objects are called here *real traces*. At least implicitly, real traces occur already in [FR82], but a systematic study began only recently and several papers are devoted to this subject, [BMP90, Kwi90, Gas90, Gas91, GR91]. One can show that there can be no convenient associative concatenation on real traces, contrary to the set of all (say countable) dependence graphs. However, since the complement of real traces in the monoid of all dependence graphs forms an ideal, one can smash all non-real dependence graphs into a single zero element. In this way, Gastin obtains a monoid which consists of all real traces with only one additional zero element. In this monoid, rational and recognizable languages can be defined in a natural way, [Gas90]. These families are closed under the usual operations, but similarly to the case of finite traces, recognizable sets are not closed neither under star-iteration nor under ω-iteration. In [GPZ91] Gastin, Petit and Zielonka gave sufficient conditions which ensure the recognizability of T^* and T^ω. However, substantially they used some finiteness properties due to simplifications by the concatenation with zero.

One of the main drawback of this concatenation is that $a^\omega a = 0 \neq$

a^ω, contrary to the word case. Diekert proposed another solution to the problem of concatenation and defined the notion of complex trace [Die91]. A possible way to see a complex trace is to start with an arbitrary dependence graph. From this graph we remember in a first component its maximal real prefix and in a second component we remember those letters which depend on actions occurring either infinitely often or in the transfinite part of the graph. Thus, a complex trace is simply a real trace together with a second component which is some finite alphabetic information. Therefore, complex traces are slightly less abstract than real traces. The advantage is that the concatenation and ω-iteration are fully defined for complex traces. In particular we have $a^\omega a = a^\omega$ for all letters a. Furthermore, in the special case of a full (empty respectively) dependence relation we just reobtain the usual construct of finite or infinite words (vectors respectively).

The aim of this paper is to define in a proper way rational and recognizable complex trace languages and to investigate their basic properties. We show various closure properties such as closure under boolean operations (for recognizable languages), concatenation, and left and right quotients by recognizable sets. Then we give sufficient conditions ensuring the recognizability of T^* and T^ω. Contrary to the concatenation with zero, in the complex calculation of T^* (T^ω respectively) an unbounded (infinite respectively) number of factors of infinite traces may be relevant. Therefore, we have to develop new techniques, which help to analyze these iterated products. In a next step we introduce the concurrent iterations c-$*$ and c-ω for complex trace languages. For this we define connected components of complex traces and the c-iterations are the usual iterations over all connected components of elements of a given language. This leads to the family of c-rational complex trace languages. Our main result is the generalization of Kleene's and Ochmanski's theorems to complex trace languages. It states that c-rational languages are the same as recognizable languages. This result is a proper generalization of Kleene's theorem for finite words, Büchi's theorem for infinite words and Ochmanski's theorem for finite traces.

The paper is organized as follows: In Section 1 we recall some definitions and facts about dependence graphs, real and complex traces, and

rational and recognizable languages. The section on rational and recognizable real trace languages (Section 2) contains all basic material which is crucial for the following. This section contains new results as well as previously known results where, however, different proofs are presented in most cases. The corresponding section for complex trace languages (Section 3) generalizes the results on real traces to complex traces. In the final section (Section 4) we define connected components of complex traces and the concurrent iteration which leads to our main result.

An Extended Abstract of a preliminary version of this paper appeared at MFCS'91, [DGP91].

1 Preliminaries

1.1 Dependence graphs

Let (X, D) be a finite *dependence alphabet*, i.e., X is a finite alphabet with a reflexive and symmetric relation $D \subseteq X \times X$ which is called the *dependence relation*. The complement $I = X \times X \setminus D$ is called the *independence relation*. The quotient monoid $\mathbb{M}(X, D) = X^*/\{ab = ba \mid (a, b) \in I\}$ is the monoid of (finite) *traces*.

It is possible to introduce infinite traces using an equivalence relation on infinite words, the definition of infinite traces is however more natural using dependence graphs. A *dependence graph* (over (X, D)) is (an isomorphism class of) a labeled acyclic graph $[V, E, \lambda]$ where V is a countable set of vertices, $E \subseteq V \times V$ is the set of arcs, $\lambda : V \to X$ is the labeling and it holds

- edges are between dependent vertices : $\forall x, y \in V$,

$$(\lambda(x), \lambda(y)) \in D \Longleftrightarrow x = y \text{ or } (x, y) \in E \text{ or } (y, x) \in E$$

- the induced partial order (V, E^*) is well-founded : there does not exist an infinite sequence $(x_i) \subseteq V$ such that $(x_{i+1}, x_i) \in E$ for all i.

The set of dependence graphs is denoted by $\mathbb{G}(X, D)$. Note that, in a dependence graph, any subset of vertices with the same label is

well-ordered. This allows to think of dependence graphs by standard representation where the vertices are pairs (a, i) with $a \in X$ and i is a countable ordinal. The restriction to countable sets is not essential here, we simply need any upper bound on the cardinality in order to stay inside set-theory.

For $g = [V, E, \lambda] \in \mathbb{G}(X, D)$ the cardinality of V is called the length of g, denoted by $|g|$. For $U \subseteq V$, let $U\downarrow = \{p \in V \mid (p, p') \in E^*$ for some $p' \in U\}$ be the downward closure of U. The restriction of g to $U\downarrow$ is a dependence graph. The set of minimal elements of g is $\min(g) = \{p \in V \mid p\downarrow = \{p\}\}$. Since $\min(g) = \min(g)\downarrow$, it may be viewed as a dependence graph which is finite since the alphabet is finite.

The set $\mathbb{G}(X, D)$ is a monoid by the concatenation $[V_1, E_1, \lambda_1]\cdot[V_2, E_2, \lambda_2] = [V, E, \lambda]$, where $[V, E, \lambda]$ is the disjoint union of $[V_1, E_1, \lambda_1]$ and $[V_2, E_2, \lambda_2]$ together with new arcs (p_1, p_2) for all $p_1 \in V_1$, $p_2 \in V_2$ such that $(\lambda_1(p_1), \lambda_2(p_2)) \in D$. The neutral element is the empty graph $1 = [\emptyset, \emptyset, \emptyset]$.

The concatenation generalizes immediately to an infinite product. Let (g_i) be any sequence of dependence graphs, then $g = g_1 g_2 \ldots \in \mathbb{G}(X, D)$ is defined as the disjoint union of the graphs (g_i) with new arcs from vertices of g_i to vertices of g_j whenever $i < j$ and the vertices have dependent labels. Thus, for any set $L \subseteq \mathbb{G}(X, D)$, the ω-iteration $L^\omega = \{g_1 g_2 \ldots \mid g_i \in L$ for $i \geq 1\} \subseteq \mathbb{G}(X, D)$ is defined. (In fact, we can define in the same way L^α for any countable ordinal α.)

Using these products, we define the canonical mapping φ from the free monoid of finite and infinite words X^∞ into the monoid of dependence graphs $\mathbb{G}(X, D)$ by $\varphi(a) = [\{p\}, \emptyset, p \mapsto a]$ for all $a \in X$ and $\varphi(a_1 a_2 \ldots) = \varphi(a_1)\varphi(a_2)\ldots$ for all word $a_1 a_2 \ldots \in X^\infty$. In other words, the dependence graph $\varphi(a_1 a_2 \ldots)$ has vertices $p_1, p_2, \ldots$ labeled by $a_1, a_2, \ldots$ and edges from p_i to p_j if and only if $i < j$ and $(a_i, a_j) \in D$. The congruence induced by φ on X^* is exactly the congruence generated by $\{ab = ba \mid (a, b) \in I\}$. Therefore, the monoid of finite traces $\mathbb{M}(X, D)$ can be identified with the submonoid of finite dependence graphs and we have $\mathbb{M}(X, D) \subseteq \mathbb{G}(X, D)$.

The image $\varphi(X^\infty) \subseteq \mathbb{G}(X, D)$ is called the set of *real traces* and is denoted by $\mathbb{R}(X, D)$. The set of real traces can be characterized as those

dependence graphs where every vertex p has a finite downward closure $p{\downarrow}$. This is a proper subset of $\mathbb{G}(X,D)$.

The main disadvantage of $\mathbb{R}(X,D)$ is that there is a convenient notion of concatenation only if D is transitive, i.e., when $\mathbb{M}(X,D) = X_1^* \times \cdots \times X_k^*$ is a direct product of free monoids. In this case, $\mathbb{R}(X,D) = X_1^\infty \times \cdots \times X_k^\infty$. However, even in this case, the concatenation will be different from the (natural) concatenation of $\mathbb{G}(X,D)$. In fact, (for $X \neq \emptyset$) the set $\mathbb{R}(X,D)$ is never a submonoid of $\mathbb{G}(X,D)$. Consider any pair $(a,b) \in D$. Then $a^\omega b \in \mathbb{G}(X,D)$ is not real, since the vertex with label b depends on infinitely many vertices. In particular, the canonical mapping $\varphi : X^\infty \to \mathbb{G}(X,D)$ is not a morphism. It neither commutes with the concatenation nor with the ω-iteration, in general. More precisely, let $L, K \subseteq X^\infty$ then we have $\varphi(L \cdot K) \neq \varphi(L) \cdot \varphi(K)$ and $\varphi(L^\omega) \neq \varphi(L)^\omega$ as soon as L contains an infinite word and K contains a nonempty word.

1.2 Complex traces

Every dependence graph $g = [V, E, \lambda] \in \mathbb{G}(X,D)$ splits into its *real part* $Re(g) = \{p \in V \mid p{\downarrow} \text{ is finite}\}$ and its *transfinite part* $Tr(g) = \{p \in V \mid p{\downarrow}$ is infinite$\}$. Of course, $Re(g)$, $Tr(g)$ are viewed as dependence graphs by restricting E and λ correspondingly. Note that for all $g \in \mathbb{G}(X,D)$ it holds $g = Re(g) \cdot Tr(g)$. The mapping $Re : \mathbb{G}(X,D) \to \mathbb{R}(X,D)$ does not define a congruence of $\mathbb{G}(X,D)$. However, we can define the coarsest congruence of $\mathbb{G}(X,D)$ such that any two congruent dependence graphs g and g' verify $Re(g) = Re(g')$. Dependence graphs falling in the same congruence class are called *practically undistinguishable* and the quotient of $\mathbb{G}(X,D)$ by this congruence is the monoid of *complex traces* $\mathbb{C}(X,D)$, [Die91].

In order to give an explicit description of $\mathbb{C}(X,D)$ we need a few more notations. For a dependence graph $g \in \mathbb{G}(X,D)$ the *alphabet* of g, denoted by $\mathrm{alph}(g)$, is the set $\lambda^{-1}(V)$. Hence, $\mathrm{alph}(g)$ is the set of letters occurring in g. The *alphabet at infinity* of g denoted by $\mathrm{alphinf}(g)$, is the set of letters occurring infinitely often in g together with $\mathrm{alph}(Tr(g))$. For a subset $A \subseteq X$ we denote the set of letters depending on A by $D(A) = \{b \in X \mid \exists a \in A : (a,b) \in D\}$, and the set of letters independent

of A by $I(A) = X \setminus D(A) = \{b \in X \mid \forall a \in A : (a,b) \in I\}$. Finally, for a dependence graph $g \in \mathbb{G}(X,D)$, we define its *imaginary part* $Im(g)$ by $Im(g) = D(\mathrm{alphinf}(g))$.

Using these notations, the coarsest congruence of $\mathbb{G}(X,D)$ which respects real parts admits the following characterization. Two dependence graphs g and g' are congruent (or practically undistinguishable) if and only if $Re(g) = Re(g')$ and $Im(g) = Im(g')$. Therefore, a complex trace is a pair $(Re(g), Im(g))$ for some dependence graph $g \in \mathbb{G}(X,D)$. Often, we denote a complex trace by $(r, D(A))$ where $r \in \mathbb{R}(X,D)$ and $A \subseteq X$. Note that the real part r is uniquely defined by the complex trace whereas the subset $A \subseteq X$ is not known, in general. (It is only $D(A)$ which is given by the complex trace.)

The concatenation of $\mathbb{C}(X,D)$ is inherited from $\mathbb{G}(X,D)$, but of course it is convenient to have an explicit formula. For this purpose, we introduce the μ-notation. Let $g \in \mathbb{G}(X,D)$ be a dependence graph and $A \subseteq X$ be any subset. Then $\mu_A(g) \in \mathbb{R}(X,D)$ is the maximal real prefix of g containing letters from $I(A)$, only. Thus, for $g = [V,E,\lambda]$, the prefix $\mu_A(g)$ is the restriction of g to $\{p \in V \mid p{\downarrow}$ is finite and $\mathrm{alph}(p{\downarrow}) \subseteq I(A)\}$. Note that, for $g \in \mathbb{G}(X,D)$ and $A \subseteq X$, we have $\mu_A(g) = \mu_A(Re(g))$. Hence the μ-notation is also well-defined for complex traces. Since $\mathbb{G}(X,D)$ is left-cancellative, [Die91], there is a unique dependence graph $\mathrm{Suff}_A(g)$ such that $g = \mu_A(g) \cdot \mathrm{Suff}_A(g)$. With these notations the concatenation in $\mathbb{C}(X,D)$ becomes [Die91]:

$$(r, D(A)) \cdot (s, D(B)) = (r \cdot \mu_A(s), D(A \cup B \cup \mathrm{alph}(\mathrm{Suff}_A(s))))$$

This formula is of first importance for the calculus on complex traces and will be used throughout.

It turns out that the coarsest congruence on $\mathbb{G}(X,D)$ which respects real parts is also a "congruence" for the ω-product on $\mathbb{G}(X,D)$. Therefore, the ω-iteration L^ω of a complex trace language L is well-defined. Note that, following the general definition of the ω-product for dependence graphs, we have $L^\omega = L^* \cup (L \setminus \{1\})^\omega$ if $1 \in L$.

In the remainder, we identify a real trace r with the complex trace $(r, D(\mathrm{alphinf}(r))) = (r, Im(r))$. Thus, we have $\mathbb{M}(X,D) \subseteq \mathbb{R}(X,D) \subseteq \mathbb{C}(X,D)$ and $\mathbb{C}(X,D)$ is a quotient monoid of $\mathbb{G}(X,D)$. For a subset

$A \subseteq X$ and a language $L \subseteq \mathbb{C}(X,D)$ ($L \subseteq \mathbb{R}(X,D)$ respectively, $L \subseteq \mathbb{G}(X,D)$ respectively) we define $L_A = \{x \in L \mid Im(x) = D(A)\}$. Note that $\mathbb{R}(X,D)_\emptyset = \mathbb{C}(X,D)_\emptyset = \mathbb{G}(X,D)_\emptyset = \mathbb{M}(X,D)$. A language L is called *finitary* if $L = L_\emptyset$, i.e., $L \subseteq \mathbb{M}(X,D)$.

The link between real and complex trace languages can be done using concatenations by finite shift-traces. A finite trace $s_{A,B}$ is called a *shift trace* from $D(A)$ to $D(B)$ if for some complex trace $(r, D(A))$ we have $(r, D(A)) \cdot s_{A,B} = (r, D(B))$. In this case, we have $(t, D(A)) \cdot s_{A,B} = (t, D(B))$ for all complex traces $(t, D(A))$. A shift trace $s_{A,B}$ satisfies $\mu_A(s_{A,B}) = 1$ (i.e., $\min(s_{A,B}) \subseteq D(A)$ and $D(A \cup \mathrm{alph}(s_{A,B})) = D(B)$). Shift traces from $D(A)$ to $D(B)$ do not exist for all pairs (A,B). The necessary and sufficient condition is that there exists a sequence of letters $a_1, \ldots, a_k$ such that $a_i \in D(A \cup \{a_1, \ldots, a_{i-1}\})$ for $1 \leq i \leq k$ and $D(B) = D(A \cup \{a_1, \ldots, a_k\})$. We simply write A shift B in this case. If A shift B then some shift trace $s_{A,B}$ can be chosen of length at most $|D(B)|$.

It is easy to see that every complex trace language $L \subseteq \mathbb{C}(X,D)$ can be written as a finite union of real trace languages concatenated by shift traces:

$$L = \bigcup_{A \text{ shift } B} (Re(L_B) \cap \mathbb{R}(X,D)_A) \cdot s_{A,B} \tag{3}$$

Let us point out that the word monoid X^∞ with right-absorbent concatenation $x \cdot y = x$ for $x \in X^\omega$ is just the special case of the complex trace monoid $\mathbb{C}(X,D)$ for a full dependence relation $D = X \times X$. In fact, if the independence relation is empty then the imaginary part is redundant. For a word x we have $Im(x) = \emptyset$ if x is finite and $Im(x) = X$ otherwise (i.e., if x is an infinite word). Furthermore, the partition $X^\infty = X^* \cup X^\omega$ is the partition of X^∞ into $(X^\infty)_\emptyset$ and $(X^\infty)_X$.

In the remainder, we will study rational and recognizable complex trace languages. As it becomes clear from Equation (3), the overall strategy is to transfer results from real trace languages. Since $\mathbb{R}(X,D) = \varphi(X^\infty)$, a basic technique used here to obtain results on real traces is to work with representing words. The next section briefly recalls some well-known properties of word languages.

1.3 Regular languages

We have (at least) two possibilities to define regular subsets of the monoid of finite and infinite words X^∞. We can use either rational expressions or finite state acceptors. This leads to the families of rational and recognizable languages respectively.

The family of rational word languages, $\mathrm{Rat}(X^\infty)$, is defined as the smallest family of subsets of X^∞ which contains all finite languages of finite words and which is closed under the operations union, concatenation, $*$-iteration, and ω-iteration. From the definition of the concatenation in X^∞, it is clear that we may restrict the operations of concatenation $L \cdot K$, of Kleene-$*$ L^*, and of ω-iteration L^ω to the case of finitary languages $L \subseteq X^*$. Thus, we are never forced to use the right-absorbent concatenation $x \cdot y = x$ with $x \in X^\omega$.

We define recognizable word languages using non-deterministic Büchi automata. There are several equivalent possibilities to give acceptance conditions. For technical reasons we use here acceptance by transitions. This allows in particular to construct automata which accept languages from X^∞ without separation of the finitary part. Moreover, this acceptance type is very convenient for our purposes since from the set of transitions which occur infinitely often in a path, we can deduce directly the alphabet at infinity of its label. To be precise, we define a Büchi automaton to be a tuple (Q, δ, q_0, Δ) where Q is the finite set of states, $\delta \subseteq Q \times (X \cup \{1\}) \times Q$ is a finite set of labeled arcs called transitions, $q_0 \in Q$ is the initial state, and $\Delta \subseteq \delta$ is the set of final transitions. An infinite path is a sequence $q_0, x_1, q_1, x_2, q_2, \ldots$ where $(q_{i-1}, x_i, q_i) \in \delta$ for all $i > 0$ and its label is the word $x_1 x_2 \cdots \in X^\infty$. A finite or infinite word x is accepted if there exists an infinite path labeled by x, starting in q_0 and repeating some transition from Δ infinitely often. Note that, finite words can be accepted by infinite loops of 1-transitions. A language $L \subseteq X^\infty$ is called *recognizable* if it is accepted by some Büchi automaton.

Another equivalent definition for recognizable languages is to use recognizing morphisms. Let $\eta : X^* \to S$ be a morphism to a finite monoid S. We say that η *recognizes* a language $L \subseteq X^\infty$ if for any finite or infinite sequence $(x_i) \subseteq X^*$, we have $x_1 x_2 \ldots \in L$ implies

$\eta^{-1}\eta(x_1)\eta^{-1}\eta(x_2)\ldots \subseteq L$. Using Ramsey factorization, one can show that any recognizable language can be written as a finite union

$$L = \bigcup_{(s,e)\in P} X_s X_e^\omega \tag{4}$$

where $X_s = \eta^{-1}(s)$, $X_e = \eta^{-1}(e)$ and $P = \{(s,e) \in S \times S \mid se = s,\ e^2 = e,\ X_s X_e^\omega \cap L \neq \emptyset\}$.

Büchi's classical theorem asserts that for X^∞ the families of rational languages and of recognizable languages coincide. We can therefore use the term *regular* languages without any ambiguity if it refers to word languages which are rational (recognizable respectively.) A detailed study of regular word languages can be found in [PP91].

1.4 Ramsey factorization

Let $\eta : M \leftrightarrows S$ be a morphism from an arbitrary monoid M to a finite monoid S and let $u_0, u_1, \ldots$ be an infinite sequence of elements of M. There exist an infinite increasing sequence of integers $i_0 < i_1 < \ldots$ and $s, e \in S$ such that $se = s$, $e^2 = e$, $\eta(u_0 \ldots u_{i_0}) = s$ and $\eta(u_{1+i_j} \ldots u_{i_k}) = e$ for all $0 \leq j < k$. This is called a *Ramsey factorization* of the infinite sequence $u_0, u_1, \ldots$.

The existence of a Ramsey factorization is one of the fundamental principles used in this paper. A simple and direct proof of this result can be found in [PP91]. It is also a trivial application of Ramsey's Theorem [Gra81].

2 Rational and recognizable real trace languages

Recall that in a general monoid M, the family $\mathrm{Rat}(M)$ is the least family which contains the finite sets and which is closed under union, product and Kleene's iteration. This definition applies in particular for the free monoid X^* and the finitary trace monoid $\mathbb{M}(X, D)$. In order to define

the family $\mathrm{Rat}(X^\infty)$, we start with finite sets of finite words and the closure under ω-iteration is required, too. Since there is no concatenation defined for real traces in general, we use restricted operations, only.

Definition: *The family of rational real trace languages,* $\mathrm{Rat}(\mathbb{R}(X,D))$, *is the smallest family satisfying the following conditions:*

i) Every finite set of finite traces is rational.

ii) If $K, L \subseteq \mathbb{R}(X,D)$ *are rational, then the union* $K \cup L$ *is rational.*

iii) If $K, L \subseteq \mathbb{R}(X,D)$ *are rational and if in addition* $K \subseteq \mathbb{M}(X,D)$, *then* $K \cdot L$, K^*, *and* K^ω *are rational.*

This definition is consistent with the definitions of $\mathrm{Rat}(X^\infty)$ and $\mathrm{Rat}(\mathbb{M}(X,D))$ since $\mathrm{Rat}(\mathbb{R}(X,D)) = \mathrm{Rat}(X^\infty)$ when $D = X \times X$ is the full dependence relation and for $L \subseteq \mathbb{M}(X,D)$ we have $L \in \mathrm{Rat}(\mathbb{R}(X,D))$ if and only if $L \in \mathrm{Rat}(\mathbb{M}(X,D))$.

Note that the canonical mapping $\varphi : X^\infty \to \mathbb{R}(X,D)$ commutes with these restricted operations. This is used in the next proposition which gives several equivalent characterizations of the family $\mathrm{Rat}(\mathbb{R}(X,D))$.

Proposition 2.1 *Let* $L \subseteq \mathbb{R}(X,D)$ *be any real trace language. Then the following assertions are equivalent.*

i) The language L *is rational, i.e.,* $L \in \mathrm{Rat}(\mathbb{R}(X,D))$

ii) The language L *is the image of a regular word language, i.e.,* $L = \varphi(L')$ *for some* $L' \in \mathrm{Rat}(X^\infty)$.

iii) The language L *can be written as a finite union*

$$L = \bigcup_{1 \le i \le k} M_i N_i^\omega$$

where M_i, N_i *are finitary rational trace languages for all* $1 \le i \le k$.

Proof: i) $\Rightarrow$ ii): First, assume that L is a finite set of finite traces. We have $L = \varphi(\varphi^{-1}(L))$ and $\varphi^{-1}(L)$ is rational since it is finite. Now,

for $K', L' \subseteq X^\infty$ we have $\varphi(K' \cup L') = \varphi(K') \cup \varphi(L')$ and if moreover $K' \subseteq X^*$ then $\varphi(K' \cdot L') = \varphi(K') \cdot \varphi(L')$, $\varphi(K'^*) = \varphi(K')^*$ and $\varphi(K'^\omega) = \varphi(K')^\omega$. Therefore, the result follows by induction on the rational expression defining L.
ii) $\Rightarrow$ iii): Using Equation (4) we obtain $L' = \bigcup_{1 \leq i \leq k} M'_i \cdot N'^\omega_i$ where $M'_i, N'_i \in \mathrm{Rat}(X^*)$ for $1 \leq i \leq k$. Since φ commutes with these operations, we obtain $L = \bigcup_{1 \leq i \leq k} \varphi(M'_i) \cdot \varphi(N'_i)^\omega$. Using again the same property of φ we deduce that the languages $\varphi(M'_i), \varphi(N'_i)$ are finitary rational trace languages, which concludes this part.
iii) $\Rightarrow$ i): trivial. □

Recall that we have defined recognizable word languages by Büchi automata. In order to define recognizable real trace languages we will consider automata which accept closed languages. A word language $L \subseteq X^\infty$ is said to be *closed* (with respect to (X, D)) if $L = \varphi^{-1}\varphi(L)$ for the canonical mapping $\varphi : X^\infty \to \mathbb{R}(X, D)$. If a Büchi automaton accepts a closed language then there is no ambiguity in saying which real traces are accepted. Moreover we will see that it is decidable whether a Büchi automaton accepts a closed language or not. This approach leads to the following definition.

Definition: *A language $L \subseteq \mathbb{R}(X, D)$ is called* recognizable *if $\varphi^{-1}(L)$ is a (closed) regular word language. The family of recognizable real trace languages is denoted by* $\mathrm{Rec}(\mathbb{R}(X, D))$.

Remark 2.2 Since the family $\mathrm{Rec}(X^\infty)$ is a boolean algebra, it follows directly from this definition that the family of recognizable real trace languages forms a boolean algebra, too. It is closed under union, intersection, and complementation.

Another possible way to define recognizable languages is, exactly as in the word case, by *recognizing morphisms*. Let $\eta : \mathbb{M}(X, D) \to S$ be a morphism to a finite monoid S. We say that η recognizes a language $L \subseteq \mathbb{R}(X, D)$, if for any finite or infinite sequence $(x_i) \subseteq \mathbb{M}(X, D)$, $x_1 x_2 \ldots \in L$ implies $\eta^{-1}\eta(x_1)\eta^{-1}\eta(x_2) \ldots \subseteq L$. We could also use the generalization to trace languages of the syntactic congruence defined by

Arnold for word languages [Arn85]. Given $L \subseteq \mathbb{R}(X,D)$, two finite traces x, y are syntactically congruent if and only if:

$$\begin{aligned} \forall u,v \in \mathbb{M}(X,D) \quad &: \quad u(xv)^\omega \in L \Leftrightarrow u(yv)^\omega \in L \\ \forall u,v,w \in \mathbb{M}(X,D) \quad &: \quad uxvw^\omega \in L \Leftrightarrow uyvw^\omega \in L \end{aligned}$$

We denote the syntactic congruence of L by $\equiv_L$ and we obtain a canonical morphism $\eta_L : \mathbb{M}(X,D) \to \mathbb{M}(X,D)/\equiv_L$. In general, neither $\mathbb{M}(X,D)/\equiv_L$ is finite nor does η_L recognize L. This is true however if L is recognizable. We have the following proposition, stating that all approaches are equivalent.

Proposition 2.3 ([Gas91]) *A language $L \subseteq \mathbb{R}(X,D)$ is recognizable if and only if any of the following equivalent definitions is satisfied.*

i) The language L is the image of a closed regular word language.

ii) There exists a morphism η form $\mathbb{M}(X,D)$ to a finite monoid S recognizing L.

iii) The syntactic congruence $\equiv_L$ is of finite index in $\mathbb{M}(X,D)$ and the syntactic morphism η_L recognizes L.

From Propositions 2.1 and 2.3, it is clear that every recognizable language is rational. The converse holds for strings by Büchi's Theorem. However, as soon as there is a pair $(a,b) \in I$ of independent letters, we find a finitary rational language, e.g., $L = \varphi(ab)^* \subseteq \mathbb{M}(X,D)$, which is rational but not recognizable (since $\varphi^{-1}(L) = \{w \in \{a,b\}^* \mid |w|_a = |w|_b\}$ is not a recognizable word language).

An example of an ω-language $L \subseteq \mathbf{R}(X,D) \setminus \mathbb{M}(X,D)$ which is rational but not recognizable, can be constructed on three letters, say a, b, c where there is at least one pair of independent letters, say $(a,b) \in I$. Then the rational language $L = \varphi(ab)^* \cdot \varphi(c)^\omega$ is not recognizable since the projection of $\varphi^{-1}(L)$ to $\{a,b\}^\infty$ is $\{w \in \{a,b\}^* \mid |w|_a = |w|_b\}$.

We leave it as an exercise to the reader to show that no such two letters example exists.

Rational trace languages are not closed under intersection, in general. For example, let $(X,D) = a \quad b \text{ —— } c$ then $\varphi(ab)^*\varphi(c)^* \cap$

$\varphi(b)^*\varphi(ac)^* = \{\varphi(a^n b^n c^n) \mid n \geq 0\}$, which is not rational since the projection to $\{b,c\}^*$ yields the non rational language $\{b^n c^n \mid n \geq 0\} \subseteq \{b,c\}^*$. However we have the following observation. It is stated as a lemma since we will use it throughout.

Lemma 2.4 *Let $K \subseteq \mathbb{R}(X,D)$ be rational and $L \subseteq \mathbb{R}(X,D)$ be recognizable. Then the intersection $K \cap L$ is rational.*

Proof: Let $K', L' \subseteq X^\infty$ be regular word languages such that $\varphi(K') = K$, $\varphi(L') = L$ and L' is closed. Then $K' \cap L'$ is a regular word language and we have $\varphi(K' \cap L') = K \cap L$. Hence, by Proposition 2.1, $K \cap L$ is rational. □

The following combinatorial lemma becomes important below.

Lemma 2.5 *Let $t = t_1 t_2 \ldots$, $z = z_1 z_2 \ldots \in X^\infty$ be two factorizations for $t, z \in X^\infty$ with $t_i, z_i \in X^*$. Then we have $\varphi(t) = \varphi(z) \in \mathbb{R}(X,D)$ if and only if there are sequences of integers $0 = m_0 < m_1 < m_2 < \ldots$, $0 = n_0 < n_1 < n_2 < \ldots$, and finite words $1 = y_0, x_1, y_1, x_2, y_2, \ldots \in X^*$ such that it holds for all $i \geq 1$:*

$$\begin{aligned} \varphi(y_{i-1}x_i) &= \varphi(t_{1+m_{i-1}} \ldots t_{m_i}) \\ \varphi(x_i y_i) &= \varphi(z_{1+n_{i-1}} \ldots z_{n_i}) \end{aligned}$$

A rough picture of the situation described in the lemma above is given in the following picture.

$$\begin{array}{lccccc} \varphi(t) & = & \varphi(t_1 \cdots t_{m_1}) & \varphi(t_{1+m_1} \cdots t_{m_2}) & \varphi(t_{1+m_2} \cdots t_{m_3}) & \ldots \\ & = & \varphi(x_1) & \varphi(y_1 x_2) & \varphi(y_2 x_3) & \ldots \\ & = & \varphi(x_1 y_1) & \varphi(x_2 y_2) & \varphi(x_3 y_3) & \ldots \\ & = & \varphi(z_1 \cdots z_{n_1}) & \varphi(z_{1+n_1} \cdots z_{n_2}) & \varphi(z_{1+n_2} \cdots z_{n_3}) & \ldots \\ & = & \varphi(z) & & & \end{array}$$

Double factorization of two words representing the same real trace

Proof: It is clear that the conditions imply $\varphi(t) = \varphi(y_0 x_1 y_1 x_2 y_2 \ldots) = \varphi(z)$. Therefore it is enough to show the other direction and we may assume $\varphi(t) = \varphi(z)$. Let $y_0 = 1$, $x_1 = t_1$ and $m_1 = 1$. Then we may assume

by induction that for some $k \geq 1$ we found indices $0 = m_0 < \ldots < m_k$ and $0 = n_0 < \ldots < n_{k-1}$ and finite words $1 = y_0, x_1, \ldots, y_{k-1}, x_k$ such that

$$\begin{aligned} \varphi(y_{i-1}x_i) &= \varphi(t_{1+m_{i-1}} \ldots t_{m_i}) \quad \text{for } 1 \leq i \leq k \\ \varphi(x_i y_i) &= \varphi(z_{1+n_{i-1}} \ldots z_{n_i}) \quad \text{for } 1 \leq i \leq k \Leftrightarrow 1. \end{aligned}$$

Since $\varphi(y_0 x_1 \ldots y_{k-1}x_k)$ is a finite prefix of $\varphi(t) = \varphi(z)$ and $\varphi(y_0 x_1 \ldots x_{k-1}y_{k-1}) = \varphi(z_1 \ldots z_{n_{k-1}})$ it follows from the left-cancellativity of $\mathbf{R}(X, D)$ that $\varphi(x_k)$ is a finite prefix of $\varphi(z_{1+n_{k-1}} z_{2+n_{k-1}} \ldots)$. Hence we find some $n_k > n_{k-1}$ such that $\varphi(x_k)$ is a prefix of $\varphi(z_{1+n_{k-1}} \ldots z_{n_k})$, i.e., $\varphi(x_k y_k) = \varphi(z_{1+n_{k-1}} \ldots z_{n_k})$ for some $y_k \in X^*$. A symmetric argument yields that for some $m_{k+1} > m_k$ there exists $x_{k+1} \in X^*$ such that $\varphi(y_k x_{k+1}) = \varphi(t_{1+m_k} \ldots t_{m_{k+1}})$. This concludes the proof of the lemma. □

A first application of the lemma above shows that it is decidable whether a regular word language is closed by computing its syntactic congruence. If a language $L \subseteq X^\infty$ is closed, then obviously its syntactic congruence $\equiv_L$ verifies $ab \equiv_L ba$ for all $(a, b) \in I$. The converse is easy for finitary languages. The next theorem states that the converse is true, in general. The decidability result follows since the syntactic congruence of a regular word language is computable [PP91].

Theorem 2.6 *A regular language $L \subseteq X^\infty$ is closed (i.e., $L = \varphi^{-1}\varphi(L)$) if and only if we have $ab \equiv_L ba$ for all $(a, b) \in I$.*

Proof: We have to show that $ab \equiv_L ba$ for all $(a, b) \in I$ implies $L = \varphi^{-1}\varphi(L)$. Let $t \in L$ and $z \in X^\infty$ such that $\varphi(t) = \varphi(z)$. Applying Lemma 2.5 to the trivial factorizations $t = a_1 a_2 \ldots$ and $z = b_1 b_2 \ldots$ with $a_i, b_i \in X \cup \{1\}$, we find two new factorizations $t = t_1 t_2 \ldots$ and $z = z_1 z_2 \ldots$ with $t_i, z_i \in X^*$ and a sequence of finite words $1 = y_0, x_1, y_1, x_2, y_2, \ldots \in X^*$ such that $\varphi(y_{i-1}x_i) = \varphi(t_i)$ and $\varphi(x_i y_i) = \varphi(z_i)$ for $i \geq 1$.

Let $\eta_L : X^* \to X^*/\equiv_L$ be the syntactic morphism of L. Since $ab \equiv_L ba$ for all $(a, b) \in I$ we have $\eta(t_i) = \eta(y_{i-1}x_i)$ for $i \geq 1$. Since

η recognizes L this implies $y_0x_1y_1x_2\ldots \in L$. In the same way, $\eta(z_i) = \eta(x_iy_i)$ for $i \geq 1$, whence $z \in L$. $\square$

However, the main interest in Lemma 2.5 results from the fact that it yields a simplified proof for the following result.

Theorem 2.7 ([GPZ91]) *Let $L \subseteq \mathbf{M}(X, D)$ be a finitary recognizable trace language such that $L = L^+$. Then $L^\omega \subseteq \mathbf{R}(X, D)$ is recognizable, too.*

Proof: Let (Q, δ, q_0, F) be a classical finite deterministic word automaton recognizing $\varphi^{-1}(L)$ by final states, i.e., $F \subseteq Q$ and a finite path is accepted if its initial state is q_0 and its final state is in F.

We now construct a Büchi automaton recognizing $\varphi^{-1}(L^\omega)$ as follows. The set of states is $Q \times Q \times \mathcal{P}(X)$ and the initial state is $(q_0, q_0, \emptyset)$. For each letter $a \in X$ and state $(q, q', A) \in Q \times Q \times \mathcal{P}(X)$ we define the transition

$$(q, q', A) \overset{a}{\Leftrightarrow\!\!\!\rightarrow} (q, \delta(q', a), A \cup \{a\})$$

and if $a \in I(A)$ then additionally another transition

$$(q, q', A) \overset{a}{\Leftrightarrow\!\!\!\rightarrow} (\delta(q, a), q', A)$$

Furthermore, if $q \in F$, we add a final 1-transition:

$$(q, q', A) \overset{1}{\Leftrightarrow\!\!\!\rightarrow} (q', q_0, \emptyset).$$

Such a non-deterministic jump signifies that a member of $\varphi^{-1}(L)$ has been computed in the first component.

Note that, if we have a path $(p, p', A) \overset{*}{\Leftrightarrow\!\!\!\rightarrow}^{z} (q, q', B)$ without 1-transition in this automaton then there exist $x, y \in X^*$ such that $\varphi(z) = \varphi(xy)$, $q = \delta(p, x)$, $q' = \delta(p', y)$ and $B = A \cup alph(y)$. (The converse is trivially true.)

According to the definition of Section 1.3 this automaton accepts all finite or infinite words which are labelings of infinite paths using infinitely many final 1-transitions. By definition, the accepted language is recognizable. Thus, we have to show only that the accepted language is $\varphi^{-1}(L^\omega)$.

First, let $z \in X^\infty$ be an accepted word. Marking the final transitions in its accepted path, we obtain the factorization:

$$\begin{array}{lll} & (q_0, q_0, \emptyset) & \overset{*}{\Leftrightarrow}\!\!\rightarrow^{z_1} \quad (q_1, q_1', A_1) \\ \overset{1}{\Leftrightarrow}\!\!\rightarrow & (q_1', q_0, \emptyset) & \overset{*}{\Leftrightarrow}\!\!\rightarrow^{z_2} \quad (q_2, q_2', A_2) \\ \overset{1}{\Leftrightarrow}\!\!\rightarrow & (q_2', q_0, \emptyset) & \overset{*}{\Leftrightarrow}\!\!\rightarrow^{z_3} \quad \dots \end{array}$$

with $z = z_1 z_2 z_3 \dots$.

Using the remark above, we find a sequence or finite words $x_1, y_1, x_2, y_2, \dots \in X^*$ such that, with $q_0' = q_0$, we have $\varphi(z_i) = \varphi(x_i y_i)$, $q_i = \delta(q_{i-1}', x_i)$ and $q_i' = \delta(q_0, y_i)$ for all $i \geq 1$. Thus, with $y_0 = 1$, we obtain for all $i \geq 1$ $\delta(q_0, y_{i-1} x_i) = q_i \in F$, i.e., $y_{i-1} x_i \in \varphi^{-1}(L)$. Hence, we have $\varphi(z) = \varphi(x_1 y_1)\varphi(x_2 y_2) \dots = \varphi(y_0 x_1)\varphi(y_1 x_2) \dots \in L^\omega$.

For the other direction let $z = b_1 b_2 \dots \in X^\infty$ be any word in $\varphi^{-1}(L^\omega)$. Then we have $\varphi(z) = \varphi(t)$ for some $t = t_1 t_2 \dots \in X^\infty$ with $t_i \in \varphi^{-1}(L)$ for all $i \geq 1$. Using Lemma 2.5, we obtain a sequence of finite words $1 = y_0, x_1, y_1, x_2, y_2, \dots \in X^*$ and two factorizations $z = z_1 z_2 \dots$ and $t = (t_1 \dots t_{m_1})(t_{1+m_1} \dots t_{m_2}) \dots$ such that for all $i \geq 1$ we have

$$\varphi(z_i) = \varphi(x_i y_i) \text{ and } \varphi(y_{i-1} x_i) = \varphi(t_{1+m_{i-1}} \dots t_{m_i}) \in L^* = L$$

Therefore, we find an accepting path for z as follows:

$$\begin{array}{lll} & (q_0, q_0, \emptyset) & \overset{*}{\Leftrightarrow}\!\!\rightarrow^{z_1} \quad (\delta(q_0, x_1), \delta(q_0, y_1), \mathrm{alph}(y_1)) \\ \overset{1}{\Leftrightarrow}\!\!\rightarrow & (\delta(q_0, y_1), q_0, \emptyset) & \overset{*}{\Leftrightarrow}\!\!\rightarrow^{z_2} \quad (\delta(q_0, y_1 x_2), \delta(q_0, y_2), \mathrm{alph}(y_2)) \\ \overset{1}{\Leftrightarrow}\!\!\rightarrow & (\delta(q_0, y_2), q_0, \emptyset) & \overset{*}{\Leftrightarrow}\!\!\rightarrow^{z_3} \quad \dots \end{array}$$

□

The main application of the theorem above is when L is connected. A finite trace is called *connected* if it is connected as a dependence graph or what is the same if its alphabet induces a connected subgraph of (X, D). A finitary language is called *connected* if all its elements are connected. Due to independent works by Métivier [Mét86] and Ochmanski [Och85], it is known that L^* (and hence L^+) is recognizable if L is recognizable and connected. Therefore we have the following corollary.

Corollary 2.8 *Let $L \subseteq \mathbb{M}(X,D)$ be recognizable and connected. Then L^ω is recognizable.*

Proof: We have $L^\omega = (L^+)^\omega$. □

Note that in $\mathbb{G}(X,D)$ the concatenation xy of two real traces is real if and only if $\mathrm{alphinf}(x) \times \mathrm{alph}(y) \subseteq I$. Therefore, even if the concatenation $K \cdot L$, as a rational operation on real trace languages, has been defined only when K is finitary, it can clearly be extended to the case where $\mathrm{alphinf}(x) \times \mathrm{alph}(y) \subseteq I$ for all $x \in K$ and $y \in L$. The reason that the rational operation was defined in a more restricted way was to maintain commutation with the canonical mapping $\varphi : X^\infty \to \mathbb{R}(X,D)$. This property is not important for the closure under concatenation of $\mathrm{Rec}(\mathbb{R}(X,D))$. It was proved by Gastin [Gas91] that the family of recognizable real trace languages is closed under this extension of the real concatenation. The key to the proof of this result is the following lemma which introduces the I-shuffle and for which we give a new proof using non-deterministic Büchi automata.

For the sake of simplicity, we write $(x,y) \in I$ if two words or real traces x and y are fully independent, i.e., if $\mathrm{alph}(x) \times \mathrm{alph}(y) \subseteq I$.

Lemma 2.9 *Let $K', L' \subseteq X^\infty$ be regular. Then the following I-shuffle is regular, too.*

$$K' \sqcup\!\sqcup_I L' = \{u_1 v_1 u_2 v_2 \ldots \in X^\infty \mid u_1 u_2 \ldots \in K', v_1 v_2 \ldots \in L' \text{ and } (v_i, u_j) \in I \text{ for all } i < j\}$$

Proof: For $i = 1, 2$, let $\mathcal{A}_i = (Q_i, \delta_i, q_{i,0}, \Delta_i)$ be Büchi automata accepting K' and L' respectively. We may assume that for $i = 1, 2$, $(q_i, 1, q_i) \in \delta_i$ for all $q_i \in Q_i$. We construct an automaton $\mathcal{A}$ for $K' \sqcup\!\sqcup_I L'$ as follows. The set of states is $Q_1 \times Q_2 \times \mathcal{P}(X) \times \mathcal{P}(\{1,2\})$ and the initial state is $(q_{1,0}, q_{2,0}, \emptyset, \emptyset)$. We define simultaneously the transitions of $\mathcal{A}$ and their projections on $\mathcal{A}_1$ and $\mathcal{A}_2$.

- If $t_1 = (q_1, x, q_1') \in \delta_1$ and $\mathrm{alph}(x) \times A \subseteq I$ then

$$(q_1, q_2, A, B) \overset{x}{\Leftrightarrow} (q_1', q_2, A, B')$$

with $B' = B \cup \{1\}$ if $t_1 \in \Delta_1$ and $B' = B$ otherwise.

Moreover, the projections of this transition are t_1 and $(q_2, 1, q_2)$ respectively.

- If $t_2 = (q_2, x, q_2') \in \delta_2$ then

$$(q_1, q_2, A, B) \overset{x}{\Leftrightarrow} (q_1, q_2', A \cup \mathrm{alph}(x), B')$$

with $B' = B \cup \{2\}$ if $t_2 \in \Delta_2$ and $B' = B$ otherwise.

Moreover, the projections of this transition are $(q_1, 1, q_1)$ and t_2 respectively.

- Finally, we add the final 1-transitions

$$(q_1, q_2, A, \{1, 2\}) \overset{1}{\Leftrightarrow} (q_1, q_2, A, \emptyset)$$

whose projections are $(q_1, 1, q_1)$ and $(q_2, 1, q_2)$ respectively.

Let us show that the language accepted by this automaton is $K' \sqcup\!\sqcup_I L'$. Let P be an accepted path of $\mathcal{A}$ with label w. Clearly, the projections P_1 and P_2 of P are paths of $\mathcal{A}_1$ and $\mathcal{A}_2$. Let u and v be the labels of P_1 and P_2. From the definition of the transitions, we can easily verify that $w \in u \sqcup\!\sqcup_I v$. Moreover, between two final transitions of P there must be at least one final transition in the corresponding parts of P_1 and P_2. Therefore, P_1 and P_2 are accepted paths and $w \in K' \sqcup\!\sqcup_I L'$.

Conversely, let $w \in K' \sqcup\!\sqcup_I L'$. We have $w = u_1 v_1 u_2 v_2 \ldots$ with $u = u_1 u_2 \ldots \in K'$, $v = v_1 v_2 \ldots \in L'$ and $(v_i, u_j) \in I$ for all $i < j$. Let

$$q_{1,0} = p_0 \overset{*}{\Leftrightarrow}^{u_1} p_1 \overset{*}{\Leftrightarrow}^{u_2} p_2 \overset{*}{\Leftrightarrow}^{u_3} p_3 \ldots$$

be an accepting path in $\mathcal{A}_1$ for u and let

$$q_{2,0} = q_0 \overset{*}{\Leftrightarrow}^{v_1} q_1 \overset{*}{\Leftrightarrow}^{v_2} q_2 \overset{*}{\Leftrightarrow}^{v_3} q_3 \ldots$$

be an accepting path in $\mathcal{A}_2$ for v. We construct an accepting path for w in $\mathcal{A}$ as follows:

$$\begin{array}{ll}
 & (p_0, q_0, \emptyset, \emptyset) \\
\overset{*}{\Leftrightarrow}^{v_1} & (p_1, q_1, \mathrm{alph}(v_1), B_1') \\
\overset{*}{\Leftrightarrow}^{u_2} & (p_2, q_1, \mathrm{alph}(v_1), B_2) \\
\overset{1}{\Leftrightarrow} & (p_2, q_2, \mathrm{alph}(v_1v_2), B_2'') \\
\overset{*}{\Leftrightarrow}^{v_3} & \ldots
\end{array}
\qquad
\begin{array}{ll}
\overset{*}{\Leftrightarrow}^{u_1} & (p_1, q_0, \emptyset, B_1) \\
\overset{1}{\Leftrightarrow} & (p_1, q_1, \mathrm{alph}(v_1), B_1'') \\
\overset{*}{\Leftrightarrow}^{v_2} & (p_2, q_2, \mathrm{alph}(v_1v_2), B_2') \\
\overset{*}{\Leftrightarrow}^{u_3} & (p_3, q_2, \mathrm{alph}(v_1v_2), B_3)
\end{array}$$

with for all $i \geq 1$, $B_i'' = \emptyset$ if $B_i' = \{1, 2\}$ and $B_i'' = B_i'$ otherwise. □

Proposition 2.10 ([Gas91]) *Let $K, L \subseteq \mathbf{R}(X, D)$ be recognizable (rational respectively) real trace languages such that for all $x \in K$, $y \in L$ it holds* $\mathrm{alphinf}(x) \times \mathrm{alph}(y) \subseteq I$. *Then $K \cdot L$ is recognizable (rational respectively).*

Proof: Consider the canonical mapping $\varphi : X^\infty \to \mathbf{R}(X, D)$. First, observe that for $u, v \in X^\infty$, $u \sqcup\!\sqcup_I v$ is not empty if and only if $\mathrm{alphinf}(u) \times \mathrm{alph}(v) \subseteq I$. Moreover, in this case, $\varphi(w) = \varphi(u) \cdot \varphi(v)$ for all $w \in u \sqcup\!\sqcup_I v$. Now, there exist regular languages $K', L' \subseteq X^\infty$ such that $\varphi(K') = K$ and $\varphi(L') = L$. By the observation above, we have $\varphi(K' \sqcup\!\sqcup_I L') = K \cdot L$. Hence, by Lemma 2.9 and Proposition 2.1, $K \cdot L$ is rational.

Finally, if K and L are recognizable, we may assume that $K' = \varphi^{-1}(K)$ and $L' = \varphi^{-1}(L)$ are closed (Proposition 2.3). We claim that in this case, $K' \sqcup\!\sqcup_I L'$ is closed too, i.e., $K' \sqcup\!\sqcup_I L' = \varphi^{-1}(K \cdot L)$, which proves that $K \cdot L$ is recognizable.

Let $w \in \varphi^{-1}\varphi(K' \sqcup\!\sqcup_I L') = \varphi^{-1}(K \cdot L)$ then $\varphi(w) = x \cdot y$ for some $x \in K$ and $y \in L$. We denote by w_i the i-th letter of w or 1 if $|w| < i$. For a letter a and a word (or a real trace) x we denote by $|x|_a$ the number of occurrences of a in x. We construct two sequences $u_1, v_1, u_2, v_2, \ldots \in X \cup \{1\}$ as follows: $u_i = w_i$, $v_i = 1$ if $|x|_{w_i} \geq |w_1 \ldots w_i|_{w_i}$ and $u_i = 1$, $v_i = w_i$ otherwise.

Clearly, $w = u_1v_1u_2v_2 \ldots$. Let $u = u_1u_2 \ldots$ and $v = v_1v_2 \ldots$. From $\varphi(w) = x \cdot y$ we deduce that $\varphi(u) = x$, $\varphi(v) = y$ and $(v_i, u_j) \in I$ for all

$i < j$. Hence, $u \in \varphi^{-1}(K) = K'$, $v \in \varphi^{-1}(L) = L'$ and $w \in K' \sqcup\!\sqcup_I L'$ which proves the claim. □

Remark that Proposition 2.10 states that the family $\mathrm{Rat}(\mathbb{R}(X,D))$ is closed under this extended concatenation. Hence, we may have replaced the concatenation restricted to finitary languages on the left by this extended one without changing the family $\mathrm{Rat}(\mathbb{R}(X,D))$.

As stated in Proposition 2.1, a real trace language L is rational if and only if it can be written as a finite union of sets of the form $M \cdot N^\omega$ where $M, N \subseteq \mathbb{M}(X,D)$ are rational. For recognizable languages this can be strengthened to the following normal form result.

Proposition 2.11 ([GPZ91]) *A real trace language $L \subseteq \mathbb{R}(X,D)$ is recognizable if and only if it can be written as a finite union of sets of the form $M \cdot N_1^\omega \cdots N_k^\omega = M \cdot (N_1 \cdots N_k)^\omega$ where $M, N_1, \ldots, N_k \subseteq \mathbb{M}(X,D)$ are recognizable languages such that all traces in N_i are connected and have the same alphabet A_i with $A_i \times A_j \subseteq I$ for $i \neq j$.*

Proof: First, it is clear by Remark 2.2, Corollary 2.8 and Proposition 2.10 that every such finite union is recognizable.

The original proof [GPZ91] for the other direction uses Mezei's Theorem. Here we simply use Ramsey factorization.

Let $\eta' : \mathbb{M}(X,D) \to S'$ be a morphism recognizing $L \subseteq \mathbb{R}(X,D)$. We define the direct product $S = S' \times \mathcal{P}(X)$ where the operation on the power set $\mathcal{P}(X)$ is union and the morphism $\eta : \mathbb{M}(X,D) \to S$ by $\eta(x) = (\eta'(x), \mathrm{alph}(x))$ for all $x \in \mathbb{M}(X,D)$. Note that η recognizes L, too. Moreover, for $s \in S$, the second projection $\mathrm{alph}(s)$ is well-defined and satisfies $\mathrm{alph}(x) = \mathrm{alph}(\eta(x))$ for all $x \in \mathbb{M}(X,D)$. This property will be used in the remainder of the proof.

Using Ramsey factorization, recall that we obtain [Gas91]:

$$L = \bigcup\nolimits_{(s,e)\in P} \eta^{-1}(s) \cdot \eta^{-1}(e)^\omega$$

where $P = \{(s,e) \in S \times S \mid se = s, e^2 = e \text{ and } \eta^{-1}(s) \cdot \eta^{-1}(e)^\omega \cap L \neq \emptyset\}$. For each $(s,e) \in P$, we decompose the alphabet $A = \mathrm{alph}(e)$ into

connected components, $A = \bigcup_{i=1}^{k} A_i$ such that A_i is not empty and connected and $A_i \times A_j \subseteq I$ for all $i \neq j$. Moreover, we define the finite set

$$E = \{(e_1, \ldots, e_k) \mid e = e_1 \cdots e_k \text{ and } \mathrm{alph}(e_i) = A_i, \text{ for all } 1 \leq i \leq k\}$$

We claim that

$$\eta^{-1}(s)\eta^{-1}(e)^{\omega} = \bigcup_{(e_1,\ldots,e_k)\in E} \eta^{-1}(s)\eta^{-1}(e_1)^{\omega} \cdots \eta^{-1}(e_k)^{\omega}$$

which conclude the proof. Note that, in the definition of E, we could require in addition that $se_i = s$ and $e_i^2 = e_i$ but this is not necessary for the proof of this proposition.

From the alphabetic property of η, we obtain

$$\eta^{-1}(e_1)^{\omega} \cdots \eta^{-1}(e_k)^{\omega} = (\eta^{-1}(e_1) \cdots \eta^{-1}(e_k))^{\omega}$$

Moreover, $\eta^{-1}(e_1) \cdots \eta^{-1}(e_k) \subseteq \eta^{-1}(e)$, hence one inclusion is clear.

Conversely, we define a morphism $\psi : \eta^{-1}(e) \to E$ as follows. Let x be in $\eta^{-1}(e)$. Since $\mathrm{alph}(x) = \mathrm{alph}(e) = A$, x admits a unique decomposition in connected components $x = x_1 \cdots x_k$ such that $\mathrm{alph}(x_i) = A_i$ for all $1 \leq i \leq k$. Then we set $\psi(x) = (\eta(x_1), \ldots, \eta(x_k))$.

Now, let $x_0x_1x_2 \ldots \in \eta^{-1}(s)\eta^{-1}(e)^{\omega}$ with $x_0 \in \eta^{-1}(s)$ and $x_i \in \eta^{-1}(e)$ for all $i \geq 1$. Using Ramsey factorization, we obtain an infinite sequence of integers $0 < i_1 < i_2 < i_3 < \ldots$ and a tuple $(e_1, \ldots, e_k) \in E$ such that $\psi(x_{1+i_j} \ldots x_{i_k}) = (e_1, \ldots, e_k)$ for all $1 \leq j < k$. Hence, $x_{1+i_j} \cdots x_{i_{j+1}} \in \eta^{-1}(e_1) \cdots \eta^{-1}(e_k)$ for all $j \geq 1$ and we obtain $x_{1+i_1}x_{2+i_1}x_{3+i_1} \ldots \in (\eta^{-1}(e_1) \cdots \eta^{-1}(e_k))^{\omega} = \eta^{-1}(e_1)^{\omega} \cdots \eta^{-1}(e_k)^{\omega}$. This proves the claim since $\eta(x_0x_1 \ldots x_{i_1}) = se \cdots e = se = s$. □

Corollary 2.12 *The family of recognizable real languages is the smallest family which contains finite sets of finite traces and which is closed under union, concatenation restricted to finitary languages on the left, and Kleene $*$ and ω-iteration both restricted to finitary connected languages.*

The final results of this section show the closure of rational and recognizable languages by left and right quotient. These results will become

crucial in the next section.

Theorem 2.13 *Let $K \subseteq \mathbb{R}(X,D)$ be arbitrary and $L \subseteq \mathbb{R}(X,D)$ be recognizable. Then the following left and right quotients are recognizable, too:*

$$\begin{aligned} K^{-1} \cdot L &= \{y \in \mathbb{R}(X,D) \mid \\ &\qquad xy \in L \text{ for some } x \in K \text{ with } \operatorname{alphinf}(x) \times \operatorname{alph}(y) \subseteq I\} \\ L \cdot K^{-1} &= \{x \in \mathbb{R}(X,D) \mid \\ &\qquad xy \in L \text{ for some } y \in K \text{ with } \operatorname{alphinf}(x) \times \operatorname{alph}(y) \subseteq I\} \end{aligned}$$

If, in addition, $K \subseteq \mathbb{R}(X,D)$ is recognizable, then the quotients are effectively computable.

Proof: Let $\eta : \mathbb{M}(X,D) \Longleftrightarrow S$ be a morphism onto a finite monoid S which recognizes L and let $X_s = \eta^{-1}(s)$ for $s \in S$. As in the proof of Proposition 2.11, we may also assume that $\operatorname{alph}(s)$ is well-defined for $s \in S$ and $\operatorname{alph}(x) = \operatorname{alph}(s)$ for all $x \in X_s$. For $s,t \in S$, we write $(s,t) \in I$ if $\operatorname{alph}(s) \times \operatorname{alph}(t) \subseteq I$. Since recognizable languages are closed under union, we may assume that $L = X_r X_d^\omega$ for some $r,d \in S$ with $rd = r$ and $d^2 = d$. Using Theorem 2.7 and Proposition 2.10, the result follows from the claim:

$$K^{-1} \cdot L = \bigcup_{(t,f) \in Q} X_t X_f^\omega$$

where

$$\begin{aligned} Q = \{(t,f) \in S \times S \mid\ & tf = t, f^2 = f \text{ and } X_s X_e^\omega \cap K \neq \emptyset \\ & \text{for some } (s,e) \in S \times S \\ & \text{with } se = s, e^2 = e, r = st, d = ef, \\ & \text{and } (e, tf) \in I\}. \end{aligned}$$

Note that the computation of Q is effective as soon as K is recognizable.

Let $y \in X_t X_f^\omega$ for $(t,f) \in Q$. Choose $x \in X_s X_e^\omega \cap K$ accordingly to the definition of Q. Then we have

$$xy \in X_s X_e^\omega X_t X_f^\omega = X_s X_t (X_e X_f)^\omega \subseteq X_r X_d^\omega = L$$

Hence, one inclusion of the claim is trivial.

For the other direction, let $y \in K^{-1} \cdot L$ and choose $x \in K$ such that $xy \in L$ with $\mathrm{alphinf}(x) \times \mathrm{alph}(y) \subseteq I$. Then we can write $xy = z_1 z_2 \ldots$ with $z_1 \in X_r$ and $z_i \in X_d$ for $i \geq 2$. Hence, there exist $x_1, x_2, \ldots, y_1, y_2, \ldots$ such that $z_i = x_i y_i$ for $i \geq 1, x = x_1 x_2 \ldots, y = y_1 y_2 \ldots$, and $(y_i, x_j) \in I$ for $i < j$. Ramsey factorization applied to the sequence $x = x_1 x_2 \ldots$ shows that we may assume $x_1 \in X_s, x_i \in X_e$ for $i \geq 2$ and some $s, e \in S$ such that $se = s$ and $e^2 = e$. In the next step, applying the same argument to the sequence $y = y_1 y_2 \ldots$, we may assume $y_1 \in X_t, y_i \in X_f$ for $i \geq 2$ and some $t, f \in S$ with $tf = t$ and $f^2 = f$. Since $\mathrm{alphinf}(x) = \mathrm{alph}(e)$ and $\mathrm{alph}(y) = \mathrm{alph}(tf)$, we have $(e, tf) \in I$. Hence $(t, f) \in Q$. This shows the other inclusion. The proof for $L \cdot K^{-1}$ is symmetric. □

For rational languages such a general result does not hold. We have the following counter example.

Example: Let $\mathbb{M}(X, D) = \{a, b\}^* \times \{c, d\}^* \times \{e\}^*$ be a direct product of three free monoids. Consider the following two rational sets: $K = (ae)^* b$ and $L = (ac)^* b (de)^*$. Then we have $K^{-1}L \subseteq \{c, d\}^* \times \{e\}^*$. The projection of $K^{-1}L$ to $\{c, d\}^*$ yields the non-rational set $\{w \in \{c, d\}^* \mid |w|_c \leq |w|_d\}$. In order to have an example of ω- languages, simply substitute b by b^ω.

However, the quotient of a rational language by a recognizable language is effectively rational. This is stated in the next Theorem.

Theorem 2.14 *Let $K \subseteq \mathbb{R}(X, D)$ be recognizable and $L \subseteq \mathbb{R}(X, D)$ be rational. Then the following left and right quotients are effectively computable and rational, too:*

$$\begin{aligned} K^{-1} \cdot L &= \{y \in \mathbb{R}(X, D) \mid \\ &\qquad xy \in L \text{ for some } x \in K \text{ with } \mathrm{alphinf}(x) \times \mathrm{alph}(y) \subseteq I\} \\ L \cdot K^{-1} &= \{x \in \mathbb{R}(X, D) \mid \\ &\qquad xy \in L \textit{ for some } y \in K \textit{ with } \mathrm{alphinf}(x) \times \mathrm{alph}(y) \subseteq I\} \end{aligned}$$

Proof: We give the proof for $K^{-1} \cdot L$ only. The other case $L \cdot K^{-1}$ is obtained analogously. Let $K' = \varphi^{-1}(K)$ and $L' \subseteq X^\infty$ be a regular

language such that $\varphi(L') = L$. Let X_1, X_2 be two copies of X and let $\hat{X} = X_1 \cup X_2$. Define three morphisms h, p_1, p_2 from $\hat{X}^\infty$ in X^∞ by $h(a_i) = a$, $p_i(a_i) = a$ and $p_j(a_i) = 1$ for $i, j = 1, 2$ with $i \neq j$. Finally, define the regular language $W \subseteq \hat{X}^\infty$ by its complement $\overline{W} = \cup_{(a,b)\in D} \hat{X}^* b_2 \hat{X}^* a_1 \hat{X}^\infty$.

Since regular word languages are closed under morphism, inverse morphism and intersection, the language $N' = p_2(p_1^{-1}(K') \cap h^{-1}(L') \cap W)$ is regular, too. Moreover, it is easy to verify that

$$\begin{aligned} N' &= \{v_1 v_2 \ldots \in X^\infty \mid \exists u_1 u_2 \ldots \in K' \text{ with } u_1 v_1 u_2 v_2 \ldots \in L' \\ &\qquad\qquad \text{and } (v_i, u_j) \in I \text{ for all } i < j\} \\ &= \{v \in X^\infty \mid (u \sqcup\!\sqcup_I v) \cap L' \neq \emptyset \text{ for some } u \in K'\} \end{aligned}$$

We claim that $\varphi(N') = K^{-1} \cdot L$ which proves that $K^{-1} \cdot L$ is rational.

Let v be in N'. There exists $w \in (u \sqcup\!\sqcup_I v) \cap L'$ for some $u \in K'$. In the proof of Proposition 2.10 we have shown that $\varphi(w) = \varphi(u) \cdot \varphi(v)$. Therefore, $\varphi(v) \in K^{-1} \cdot L$.

Conversely, let y be in $K^{-1} \cdot L$. There exists $x \in K$ such that $\mathrm{alphinf}(x) \times \mathrm{alph}(y) \subseteq I$ and $z = xy \in L$. Let $w \in \varphi^{-1}(z) \cap L'$. In the proof of Proposition 2.10 we have shown that $\varphi^{-1}(xy) = \varphi^{-1}(x) \sqcup\!\sqcup_I \varphi^{-1}(y)$. Hence, there exist $u \in \varphi^{-1}(x)$ and $v \in \varphi^{-1}(y)$ such that $w \in u \sqcup\!\sqcup_I v$, whence $(u \sqcup\!\sqcup_I v) \cap L' \neq \emptyset$. Since $u \in \varphi^{-1}(x) \subseteq \varphi^{-1}(K) = K'$, we obtain $v \in N'$. □

3 Rational and recognizable complex trace languages

Recall the following notations. Let $L \subseteq \mathbb{C}(X, D)$ be a complex trace language and $A \subseteq X$ be a subset of the alphabet. We denote by $L_A = \{x \in L \mid \mathrm{Im}(x) = D(A)\}$ and $\mu_A(L) = \{\mu_A(x) \mid x \in L\}$ where $\mu_A(x)$ is the maximal real prefix of $\mathrm{Re}(x)$ containing letters independent of A, only. It will be useful to have a finer partition. Let $L \subseteq \mathbb{C}(X, D)$ and $A, B \subseteq X$. We define $L_{A,B} = \{x \in L \mid \mathrm{Im}(yx) = D(B) \text{ for some } y \in \mathbb{C}(X, D)_A\}$. Note that in the definition above the existential "for some"

can be replaced by "for all". The reason to introduce $L_{A,B}$ is that if we concatenate any complex trace $y \in \mathbb{C}(X,D)_A$ with some $x \in L$ such that $yx \in \mathbb{C}(X,D)_B$, then in fact $x \in L_{A,B}$. Note that in this case $\mathrm{Re}(yx) = \mathrm{Re}(y)\mu_A(x)$, therefore, we are also interested in the language $\mu_{A,B}(L)$ which is defined by $\mu_{A,B}(L) = \mu_A(L_{A,B})$.

The formulae given in the next lemma will be useful throughout.

Lemma 3.1 *Let $K, L \subseteq \mathbb{C}(X,D)$ be complex trace languages and $B \subseteq X$. Then the following formulae hold:*

i) $L_B = L_{\emptyset,B}$

ii) $L = \bigcup_{C \subseteq X} L_{B,C}$

iii) $\mu_B(L) = \bigcup_{C \subseteq X} \mu_{B,C}(L)$

iv) $(KL)_B = \bigcup_{A \subseteq X} K_A L_{A,B}$

v) $\mathrm{Re}((KL)_B) = \bigcup_{A \subseteq X} \mathrm{Re}(K_A)\mu_{A,B}(L)$

Proof: i) is obvious. ii) Let $x \in L$ and $y \in \mathbb{C}(X,D)_B$. Then, $D(B) \subseteq \mathrm{Im}(yx) = D(C)$ for some $C \subseteq X$. Therefore $x \in L_{B,C}$. iii) follows since μ_A commutes with union. iv) We have $(KL)_B = \bigcup_{A \subseteq X}(K_A L)_B = \bigcup_{A \subseteq X}(K_A L_{A,B})$. At last, for any $x \in K_A$ and any $y \in L_{A,B}$, $\mathrm{Re}(xy) = \mathrm{Re}(x)\mu_A(y)$ and v) follows from iv). □

Finally, we define $\nu_A(L) = \mu_{A,A}(L)$. Each trace in $\nu_A(L)$ is finite since $\mathrm{alph}(u) \subseteq I(A)$ and $\mathrm{Im}(u) \subseteq D(A)$ implies $\mathrm{Im}(u) = \emptyset$. Note also that $\nu_\emptyset(L)$ is just the finitary part $L_\emptyset = L \cap \mathbb{M}(X,D)$. Later, we will introduce finite components, which could be characterized as the union of $\nu_A(L)$ over $A \subseteq X$. For the moment we are interested in the $\nu_A(L)$, because we have the following formula.

Lemma 3.2 *Let $L \subseteq \mathbb{C}(X,D)$ be a complex trace language and $\dagger \in \{*, \omega\}$. Let $\mathcal{A}$ be the finite set of sequences $\mathcal{A} = \{(A_0, \ldots, A_k) \mid \emptyset = D(A_0) \subsetneq \cdots \subsetneq D(A_k) \subseteq X\}$. Then the language $L^\dagger$ is the finite union*

$$L^\dagger = \bigcup_{(A_0,\ldots,A_k) \in \mathcal{A}} \nu_{A_0}(L)^* L_{A_0,A_1} \ldots \nu_{A_{k-1}}(L)^* L_{A_{k-1},A_k} \nu_{A_k}(L)^\dagger$$

Proof: The proof uses in a crucial way the observation that, if $x, y \in \mathbb{C}(X, D)$ and $Im(x) = Im(xy) = D(A)$ then $\mu_A(y) = \nu_A(y)$ and $xy = x\nu_A(y)$.

Let $x = x_1 x_2 \ldots \in L^\dagger$, $x_i \in L$. Eventually, the sequence of alphabets $\mathrm{Im}(x_1) \subseteq \mathrm{Im}(x_1 x_2) \subseteq \cdots$ becomes stationary. Therefore, there exist an integer k, a sequence of integers $0 = i_0 < i_1 < \ldots < i_k < i_{k+1} = \omega$ and a sequence $(A_0, \ldots, A_k) \in \mathcal{A}$ such that $Im(x_1 \ldots x_n) = D(A_j)$ for all $i_j \leq n < i_{j+1}$ and $0 \leq j \leq k$. Define now the sequence $(y_n)_{n \geq 1}$ by $y_n = \nu_{A_j}(x_n) \in \nu_{A_j}(L)$ if $i_j < n < i_{j+1}$ for some $0 \leq j \leq k$ and by $y_n = x_n \in L_{A_{j-1}, A_j}$ if $n = i_j$ for some $1 \leq j \leq k$. From the observation above, it follows $x_1 \ldots x_n = y_1 \ldots y_n$ for all n and thus $x = y_1 y_2 \ldots y_n \ldots$. Therefore,

$$x \in \nu_{A_0}(L)^* L_{A_0, A_1} \ldots \nu_{A_{k-1}}(L)^* L_{A_{k-1}, A_k} \nu_{A_k}(L)^\dagger$$

Conversely, we show by induction on k that $\nu_{A_0}(L)^* L_{A_0, A_1} \ldots \nu_{A_{k-1}}(L)^* L_{A_{k-1}, A_k} \nu_{A_k}(L)^\dagger \subseteq L^\dagger$, for any $(A_0, \ldots, A_k) \in \mathcal{A}$. For $k = 0$ this formula becomes $\nu_\emptyset(L)^\dagger = L_\emptyset^\dagger \subseteq L^\dagger$ which is clear. Assume that, for any $(A_0, \ldots, A_k) \in \mathcal{A}$, the inclusion holds and let $(A_0, \ldots, A_{k+1}) \in \mathcal{A}$. By induction hypothesis applied to $(A_0, \ldots, A_k)$, we have in particular:

$$\nu_{A_0}(L)^* L_{A_0, A_1} \ldots \nu_{A_{k-1}}(L)^* L_{A_{k-1}, A_k} \nu_{A_k}(L)^* \subseteq L^* \tag{5}$$

Since all the complex traces belonging to the left hand side of Equation (5) have clearly $D(A_k)$ as imaginary part, it follows:

$$\nu_{A_0}(L)^* L_{A_0, A_1} \ldots \nu_{A_{k-1}}(L)^* L_{A_{k-1}, A_k} \nu_{A_k}(L)^* \subseteq (L^*)_{A_k}$$

Therefore the inclusion for $(A_0, \ldots, A_{k+1})$ follows since $(L^*)_{A_k} L_{A_k, A_{k+1}} \subseteq (L^*)_{A_{k+1}}$ and, from the observation above, $(L^*)_{A_{k+1}} \nu_{A_{k+1}}(L)^\dagger \subseteq L^\dagger$ for $\dagger \in \{*, \omega\}$. $\square$

We are now ready to define rational and recognizable complex trace languages. There will be various equivalent characterizations for these notions which justify the following definition.

Definition: *A complex trace language $L \subseteq \mathbb{C}(X, D)$ is called rational (recognizable respectively) if for all $A \subseteq X$ the real language*

$\mathrm{Re}(L_A)$ *is rational (recognizable respectively). The families are denoted by* $\mathrm{Rat}(\mathbb{C}(X,D))$ *and* $\mathrm{Rec}(\mathbb{C}(X,D))$ *respectively.*

Proposition 3.3 *The family* $\mathrm{Rec}(\mathbb{C}(X,D))$ *is a boolean algebra : it is closed under union, intersection, and complementation. The family* $\mathrm{Rat}(\mathbb{C}(X,D))$ *is closed under union.*

Proof: Since $\mathrm{Re} : \mathbb{C}(X,D) \to \mathbb{R}(X,D)$ commutes with union, it follows from results of Section 2 on real traces that $\mathrm{Rat}(\mathbb{C}(X,D))$ and $\mathrm{Rec}(\mathbb{C}(X,D))$ are closed under union. The closure of $\mathrm{Rec}(\mathbb{C}(X,D))$ under intersection and complementation follows from Remark 2.2 and the formulae: for all $K, L \subseteq \mathbb{C}(X,D)$ and $A \subseteq X$, $\mathrm{Re}(K_A \cap L_A) = \mathrm{Re}(K_A) \cap \mathrm{Re}(L_A)$ and $\mathrm{Re}((\mathbb{C}(X,D) \setminus K)_A) = \mathrm{Re}(\mathbb{C}(X,D)_A) \setminus \mathrm{Re}(K_A)$. □

For any real trace language $L \subseteq \mathbb{R}(X,D)$ and any subset $A \subseteq X$, it holds $\mathrm{Re}(L_A) = \{r \in L \mid \mathrm{Im}(r)\} = D(A)\} = L \cap \mathbb{R}(X,D)_A$. Since

$$\varphi^{-1}(\mathbb{R}(X,D)_A) = \bigcup\nolimits_{\{B \subseteq X \mid D(B) = D(A)\}} \{x \in X^\infty \mid \mathrm{alphinf}(x) = B\}$$

the set $\mathbb{R}(X,D)_A$ is a recognizable real trace language. Therefore if L is a rational (recognizable respectively) real trace language, then so is $Re(L_A)$ for any $A \subseteq X$. Hence L is also a rational (recognizable respectively) complex trace language. More precisely, we have (as expected) $\mathrm{Rat}(\mathbb{R}(X,D)) = \{L \subseteq \mathbb{R}(X,D) \mid L \in \mathrm{Rat}(\mathbb{C}(X,D))\}$ and $\mathrm{Rec}(\mathbb{R}(X,D)) = \{L \subseteq \mathbb{R}(X,D) \mid L \in \mathrm{Rec}(\mathbb{C}(X,D))\}$. At last, note that since $\mathrm{Re}(L) = \bigcup_{A \subseteq X} \mathrm{Re}(L_A)$, we have also the inclusions$\mathrm{Re}(\mathrm{Rat}(\mathbb{C}(X,D)))$ $\subseteq \mathrm{Rat}(\mathbb{R}(X,D))$ and $\mathrm{Re}(\mathrm{Rec}(\mathbb{C}(X,D))) \subseteq \mathrm{Rec}(\mathbb{R}(X,D))$. The situation is summarized in the following figure.

$$\begin{array}{ccccc}
\mathrm{Rec}(\mathbb{R}(X,D)) & \subseteq & \mathrm{Rec}(\mathbb{C}(X,D)) & \overset{\mathrm{Re}}{\hookrightarrow} & \mathrm{Rec}(\mathbb{R}(X,D)) \\
\cap\mathrel{|} & & \cap\mathrel{|} & & \cap\mathrel{|} \\
\mathrm{Rat}(\mathbb{R}(X,D)) & \subseteq & \mathrm{Rat}(\mathbb{C}(X,D)) & \overset{\mathrm{Re}}{\hookrightarrow} & \mathrm{Rat}(\mathbb{R}(X,D))
\end{array}$$

Before we continue the reader should be convinced that it makes no sense to define a language $L \subseteq \mathbb{C}(X,D)$ to be rational (recognizable respectively) if the projection $\mathrm{Re}(L) \subseteq \mathbb{R}(X,D)$ is rational (recognizable

respectively). Both classes of languages would be uncountable. (Note that the families of rational and recognizable complex trace languages are countable. Indeed, a complex trace language is entirely defined by the languages L_A for $A \subseteq X$ and the family of rational real trace languages is countable since it is defined by rational expressions.) For instance, if $(X, D) = a \text{ —— } b \text{ —— } c$, for any $P \subseteq \mathbf{N}$, the language $L_P = \{(\varphi(b)^j\varphi(a)^\omega, D(a)) \mid j \in P\} \cup \{(\varphi(b)^j\varphi(a)^\omega, D(a,b)) \mid j \notin P\}$ verifies $\varphi^{-1}(\mathrm{Re}(L_P)) = a^*b^\omega$ and would be therefore recognizable. Note also that the family of recognizable languages would not be closed under intersection. Namely if $(X, D) = a \text{ —— } b \text{ —— } c \text{ —— } d$ and $L = \varphi(ac)^*\varphi(b)^\omega\varphi(c) \cup \{\varphi(a), \varphi(c)\}^*\varphi(b)^\omega$, $\mathrm{Re}(L) = \{\varphi(a), \varphi(c)\}^*\varphi(b)^\omega$ is a recognizable real trace language whereas $L_{\{b,c\}} = L \cap \mathbb{C}(X, D)_{\{b,c\}}$ is not recognizable since $\mathrm{Re}(L_{\{b,c\}}) = \varphi(ac)^*\varphi(b)^\omega$. Similarly, since the canonical mapping $\varphi : X^\infty \to \mathbb{C}(X, D)$ is not surjective, it cannot be used neither to define a language $L \subseteq \mathbb{C}(X, D)$ to be recognizable. For example, simply asking that $\varphi^{-1}(L) \subseteq X^\infty$ has to be regular would yield that any language not containing real traces is recognizable. (Of course, according to our definition, $L \in \mathit{Rec}(\mathbb{C}(X, D))$ implies that $\varphi^{-1}(L) \subseteq X^\infty$ is regular.)

Lemma 3.4 *Let $L \subseteq \mathbb{C}(X, D)$ be rational (recognizable respectively) and $A, B \subseteq X$. Then the languages $L_{A,B} \subseteq \mathbb{C}(X, D)$ and $\mu_{A,B}(L) \subseteq \mathbf{R}(X, D)$ are rational (recognizable respectively).*

Proof: First, we show the assertion for $L_{A,B}$. Note that $(L_{A,B})_C = (L_C)_{A,B}$ for all $C \subseteq X$. Thus,

$$\begin{aligned}
\mathrm{Re}((L_{A,B})_C) &= \mathrm{Re}((L_C)_{A,B}) \\
&= \mathrm{Re}(\{x \in L_C \mid \mathrm{Im}(yx) = D(B) \text{ for some } y \in \mathbb{C}(X, D)_A\}) \\
&= \{r \in \mathrm{Re}(L_C) \mid \mathrm{Im}(yr) \cup D(C) = D(B) \\
&\qquad \text{for some } y \in \mathbb{C}(X, D)_A\} \\
&= \mathrm{Re}(L_C) \cap K
\end{aligned}$$

where $K = \{r \in \mathbf{R}(X, D) \mid D(A \cup C \cup \mathrm{Suff}_A(r)) = D(B)\}$.

Using Remark 2.2 and Lemma 2.4, we are reduced to prove that the language K is recognizable. Indeed, it is easy to verify that the language

$\varphi^{-1}(K)$ is accepted by the following deterministic Büchi automaton. The set of states is $\mathcal{P}(X)$, the initial state is A and for all $q \in \mathcal{P}(X)$ and $a \in X$ we have the transition (q, a, q') where $q' = q \cup \{a\}$ if $a \in D(q)$ and $q' = q$ otherwise. A transition (q, a, q') is final if and only if $D(A \cup C \cup q) = D(B)$.

For $\mu_{A,B}(L)$ we have $\mu_{A,B}(L) = \mu_A(L_{A,B})$ by definition. Hence it is enough to show the assertion for $\mu_A(L)$. Since $\mu_A(L) = \mu_A(\mathrm{Re}(L))$, we are reduced to consider real trace language $L \subseteq \mathbb{R}(X, D)$.

For $L \subseteq \mathbb{R}(X, D)$ we have $\mu_A(L) = M \cap (L \cdot \mathrm{Suff}_A^{-1})$ where $M = \{x \in \mathbb{R}(X, D) \mid \mathrm{alph}(x) \subseteq I(A)\}$ and

$$\mathrm{Suff}_A = \{r \in \mathbf{R}(X, D) \mid \min(r) \subseteq D(A)\} = \mathbf{R}(X, D) \setminus \bigcup_{a \in I(A)} a \cdot \mathbf{R}(X, D)$$

Since M and Suff_A are recognizable, the results follow from Lemma 2.4, Theorems 2.13, and 2.14. □

So far, we have allowed as rational operation on real traces the concatenation $K \cdot L$, only if K is finitary. This was motivated by the fact that $\mathbb{R}(X, D)$ is not a monoid. For complex traces such a restriction is obviously not adequate and our first result underlines this. It follows directly from Lemma 3.1, v), Proposition 2.10 and Lemma 3.4.

Theorem 3.5 *Let $K, L \in \mathbb{C}(X, D)$ be rational (recognizable respectively). Then the concatenation $K \cdot L$ has the same property.*

We can now extend Proposition 2.1 and Proposition 2.11 and give normal forms for rational and recognizable complex trace languages.

Corollary 3.6 *i) A language $L \subseteq \mathbb{C}(X, D)$ is rational if and only if it can be written as a finite union of sets of the form $MN^\omega s$ where $M, N \subseteq \mathbb{M}(X, D)$ are finitary and rational and $s \in \mathbb{M}(X, D)$ is a finite (shift-)trace.*

ii) A language $L \subseteq \mathbb{C}(X, D)$ is recognizable if and only if it can be written as a finite union of sets of the form $MN_1^\omega \ldots N_k^\omega s$ where $M, N_1, \ldots, N_k \subseteq \mathbb{M}(X, D)$ are finitary and recognizable, each trace in N_i has the same connected alphabet A_i, $A_i \times A_j \subseteq I$ for $i \neq j$, and $s \in \mathbb{M}(X, D)$ is a finite (shift-)trace.

Proof: We have seen in the preliminaries every complex trace language $L \subseteq \mathbb{C}(X, D)$ can be written as a finite union:

$$L = \bigcup_{A \text{ shift } B} (\mathrm{Re}(L_B) \cap \mathbb{R}(X, D)_A) s_{A,B}$$

where $s_{A,B}$ is some finite shift trace from $D(A)$ to $D(B)$. Hence if L is rational (recognizable respectively), Proposition 2.1 (Proposition 2.11 respectively) applied to languages $\mathrm{Re}(L_B) \cap \mathbb{R}(X, D)_A$ yield the only-if parts of the corollary. The if-parts come from the inclusions $\mathrm{Rat}(\mathbb{R}(X, D)) \subseteq \mathrm{Rat}(\mathbb{C}(X, D))$ ($\mathrm{Rec}(\mathbb{R}(X, D)) \subseteq \mathrm{Rec}(\mathbb{C}(X, D))$ respectively) and the closures under concatenation, Theorem 3.5. □

Since the concatenation and the ω-product are fully defined in $\mathbb{C}(X, D)$, the $*$-iteration and the ω-iteration should be allowed as rational operations, in full generality. The next theorem states that this is indeed the case.

Theorem 3.7 *Let $L \in \mathrm{Rat}(\mathbb{C}(X, D))$ be a rational complex trace language. Then L^* and L^ω are rational.*

Proof: If L is a rational complex trace language, from Lemma 3.4, we deduce in particular that $\nu_A(L) \subseteq \mathbb{M}(X, D)$ is a finitary rational trace language for all $A \subseteq X$. Hence, for all $A \subseteq X$, $\nu_A(L)^*$ and $\nu_A(L)^\omega$ are rational real trace languages, too. Thus, the theorem follows from Lemma 3.2, Proposition 3.3 and Theorem 3.5. □

From closure under union, Theorems 3.5, 3.7 and Corollary 3.6, we obtain the following characterization which could also be used for a convenient definition of rational complex trace languages.

Corollary 3.8 *The family of rational complex trace languages is the smallest family which contains all finitary rational trace languages and which is closed under union, concatenation, Kleene-$*$, and ω-iteration.*

Of course, we cannot expect that recognizable languages are closed under Kleene-$*$ or ω-iteration, in general. However, let L be a recognizable complex trace language such that $\nu_A(L)^* \subseteq \mathbb{M}(X, D)$ is recognizable for all $A \subseteq X$. From Theorem 2.7, we deduce that, for all $A \subseteq X$,

$\nu_A(L)^\omega$ is recognizable, too. Hence, from Lemma 3.2 and closures under union and concatenation we obtain the following corollary. It is our strongest result concerning the question when the star- and ω-iteration of a recognizable complex trace language is recognizable.

Corollary 3.9 *Let $L \subseteq \mathbb{C}(X,D)$ be a recognizable complex trace language such that $\nu_A(L)^* \subseteq \mathbb{M}(X,D)$ is recognizable for all $A \subseteq X$. Then L^* and L^ω are recognizable complex trace languages, too.* □

We finish this section by an extension of Theorems 2.13 and 2.14 to complex trace languages.

Theorem 3.10 *Let $K \subseteq \mathbb{C}(X,D)$ be arbitrary (recognizable respectively) and $L \subseteq \mathbb{C}(X,D)$ be recognizable (rational respectively). Then the following left and right quotients are recognizable (rational respectively), too:*

$$\begin{aligned} L \cdot K^{-1} &= \{x \in \mathbb{C}(X,D) \mid xy \in L \text{ for some } y \in K\} \\ K^{-1} \cdot L &= \{y \in \mathbb{C}(X,D) \mid xy \in L \text{ for some } x \in K\} \end{aligned}$$

If K is recognizable, then the quotients are effectively computable.

Proof: First we consider $\mathrm{Re}((L \cdot K^{-1})_A)$ for some $A \subseteq X$. We have

$$(L \cdot K^{-1})_A = \bigcup_{B \subseteq X} (L_B \cdot K^{-1})_A = \bigcup_{B \subseteq X} (L_B \cdot K_{A,B}^{-1})_A$$

Hence, it remains to prove that $\mathrm{Re}((L_B \cdot K_{A,B}^{-1})_A)$ is a recognizable (rational respectively) real trace language. We have

$$\begin{aligned} \mathrm{Re}((L_B \cdot K_{A,B}^{-1})_A) &= \{r \in \mathrm{Re}(\mathbb{C}(X,D)_A) \mid r \cdot \mu_A(y) \in \mathrm{Re}(L_B) \\ &\qquad \text{for some } y \in K_{A,B}\} \\ &= \mathrm{Re}(\mathbb{C}(X,D)_A) \cap \mathrm{Re}(L_B) \cdot (\mu_{A,B}(K))^{-1} \end{aligned}$$

Hence, the results follow from Remark 2.2, Lemma 2.4, Theorems 2.13, 2.14 and Lemma 3.4.

The calculus for $K^{-1} \cdot L$ is slightly different. First, note that $K^{-1} \cdot L = \cup_{A \subseteq B \subseteq X} K_A^{-1} \cdot L_B$. Then we have

$$\begin{aligned} K_A^{-1} \cdot L_B &= \{y \in \mathbb{C}(X,D)_{A,B} \mid r\mu_A(y) \in \mathrm{Re}(L_B) \text{ for some } r \in \mathrm{Re}(K_A)\} \\ &= \mathbb{C}(X,D)_{A,B} \cap \{y \in \mathbb{C}(X,D) \mid \mu_A(y) \in \mathrm{Re}(K_A)^{-1} \cdot \mathrm{Re}(L_B)\} \end{aligned}$$

Since by Theorems 2.13 and 2.14, $\mathrm{Re}(K_A)^{-1} \cdot \mathrm{Re}(L_B)$ is recognizable (rational respectively) the results follow from the claim:

Claim: Let $R \subseteq \mathbb{R}(X,D)$ be rational (recognizable respectively) then $M = \{y \in \mathbb{C}(X,D) \mid \mu_A(y) \in R\}$ is rational (recognizable respectively).

For all $B \subseteq X$, we have $\mathrm{Re}(M_B) = \mathrm{Re}(\mathbb{C}(X,D)_B) \cap \{r \in \mathbb{R}(X,D) \mid \mu_A(r) \in R\}$. Therefore, the claim follow directly from

$$\{r \in \mathbb{R}(X,D) \mid \mu_A(r) \in R\} = (\{r \in \mathbb{R}(X,D) \mid \mathrm{alph}(r) \subseteq I(A)\} \cap R) \cdot \mathrm{Suff}_A$$

where Suff_A is the recognizable set defined in Lemma 3.4. □

In the section below we will introduce connected and finitary connected components and we will extend Kleene's and Ochmanski's theorems to complex trace languages.

4 C-rational complex trace languages

In order to define concurrent iterations for complex trace languages, we need first a notion of connectedness for complex traces. Intuitively, a trace is connected if it cannot be split into two non-empty independent traces. Thus, let us introduce the notion of independence. This notion is clear in $\mathbb{G}(X,D)$: two graphs f and g are independent if $\mathrm{alph}(f) \times \mathrm{alph}(g) \subseteq I$. Unfortunately, we cannot use this definition in $\mathbb{C}(X,D)$ since the alphabet of a complex trace is not well-defined. Nevertheless, it is easy to verify that the independence relation in $\mathbb{G}(X,D)$ factorizes to complex traces. Hence the independence relation in $\mathbb{C}(X,D)$ is well-defined and can be characterized as follows. Two complex traces $x = (u, D(A))$, $y = (v, D(B)) \in \mathbb{C}(X,D)$ are *independent* if $(\mathrm{alph}(u) \cup A) \times$

$(\mathrm{alph}(v) \cup B) \subseteq I$, in this case we will simply write $(x, y) \in I$. Note that this is well-defined since $D(B) = D(B')$ implies $(\mathrm{alph}(u) \cup A) \times (\mathrm{alph}(v) \cup B) \subseteq I \Leftrightarrow (\mathrm{alph}(u) \cup A) \cap D(\mathrm{alph}(v) \cup B) = \emptyset \Leftrightarrow (\mathrm{alph}(u) \cup A) \cap D(\mathrm{alph}(v) \cup B') = \emptyset \Leftrightarrow D(\mathrm{alph}(u) \cup A) \cap (\mathrm{alph}(v) \cup B') = \emptyset$ This leads to the following definition.

Definition: *A complex trace z is called* connected *if for all factorization $z = xy$ where x, y are independent, either $x = 1$ or $y = 1$. A trace x is called a* connected component *of $z \in \mathbb{C}(X, D)$ if either $x = z = 1$ or x is a non-empty connected trace such that $z = xy$ for some y independent of x. The trace z is said to be* finitary connected *if it admits at most one finite connected component.*

For any non-empty trace z, we will denote by $C(z)$ (respectively $\mathrm{FC}(z) = C(z) \cap \mathbb{M}(X, D)$) the set of connected components (respectively finite connected components) of z. These notations are as usual extended to languages. A language is said to be connected (respectively finitary connected) if all its elements are connected (respectively finitary connected).

Note that a direct translation of the above definition can be used to define connectedness notions on dependence graphs as well. These notions coincide with the natural notion of connectedness in $\mathbb{G}(X, D)$: a dependence graph is connected if and only if so is its underlying undirected graph. Unfortunately, this natural notion does not factorize to complex traces. The same complex trace might have connected as well as disconnected representing dependence graphs. For example let $(X, D) = a \text{ —— } b \text{ —— } c$. Then the complex trace $x = ((ac)^\omega, X)$ is represented by the connected dependence graph $(ac)^\omega b$ as well as by the disconnected graph $a^\omega c^\omega$. This explains why we define the notion of connected traces through the independence relation. For finitary connectedness the situation is simpler. A trace $x \in \mathbb{M}(X, D)$ is a finite connected component of $z \in \mathbb{C}(X, D)$ if and only if x is finite and a connected component of any representing dependence graph of z. The following proposition precises the links between the two notions of connectedness in $\mathbb{G}(X, D)$ and $\mathbb{C}(X, D)$.

Proposition 4.1 *Let $x \in \mathbb{C}(X, D)$ be a complex trace.*

i) x is connected if and only if all its representatives in $\mathbb{G}(X, D)$ are connected.

ii) Let $f \in \mathbb{G}(X, D)$ be any representative of x and let $\{f_1, \ldots, f_k\}$ be the set of finite connected components of f. Then $\mathrm{FC}(x) = \{f_1, \ldots, f_k\}$.

Proof: i) Let f be a representative of x in $\mathbb{G}(X, D)$. Suppose that $f = gh$ with $\mathrm{alph}(g) \times \mathrm{alph}(h) \subseteq I$. Let y, z be the complex traces represented by g, h respectively. Hence $x = yz$ with $(y, z) \in I$. Since x is connected, we get $y = 1$ or $z = 1$ and thus $g = 1$ or $h = 1$. Therefore f is connected.
Conversely, suppose that $x = yz$ with $(y, z) \in I$. Let g, h be representatives in $\mathbb{G}(X, D)$ of the complex traces y, z respectively. Hence gh is a representative of x and $(g, h) \in I$. Thus $g = 1$ or $h = 1$, hence $y = 1$ or $z = 1$ and x is connected.
ii) Let g be a finite connected component of f, $f = gh$ for some $h \in \mathbb{G}(X, D)$ with $\mathrm{alph}(g) \times \mathrm{alph}(h) \subseteq I$. Let y be the complex trace represented by h. Since g is finite, we get $x = gy$ with $(g, y) \in I$. Moreover g is a connected complex trace from i). Hence $g \in \mathrm{FC}(x)$.
Conversely, let $y \in \mathrm{FC}(x)$. Then $x = yz$ for some complex trace z such that $(y, z) \in I$. Since $\mathbb{G}(X, D)$ is left cancellative [Die91], the dependence graph $y^{-1}f$ is well-defined and is a representative of the complex trace z. Hence y is a finite connected component of $f = y(y^{-1}f)$. □

In order to generalize Büchi's and Ochmanski's Theorems to complex traces, we need a link to the language $\nu_A(L)$ defined above. For a complex trace x, any non-empty product over pairwise independent finite connected components of x is called a *finite component* of x. For a language $L \subseteq \mathbb{C}(X, D)$, the set of finite components of the elements of L is denoted by $\mathrm{FinComp}(L)$.

Lemma 4.2 *Let $L \subseteq \mathbb{C}(X, D)$ be recognizable. Then the following assertions hold.*

i) $\mathrm{FinComp}(L) = \bigcup_{A \subseteq X} \nu_A(L)$

ii) $\mathrm{FC}(L)$ *and* $\mathrm{C}(L)$ *are recognizable*

Proof: i) Let $t \in \nu_A(L)$. By definition, we have $\mathrm{alph}(t) \subseteq I(A)$ and $\mathrm{Im}(t) \subseteq D(A)$, hence $\mathrm{Im}(t) = \emptyset$ and thus t is finite. Moreover, there exists some $x \in L_{A,A}$ such that $t = \mu_A(x)$. Hence $x = t \cdot (r, \mathrm{Im}(x))$ and, since $x \in L_{A,A}$, we have $(D(\mathrm{alph}(r)) \cup \mathrm{Im}(x)) \subseteq D(A)$. Thus, from $\mathrm{alph}(t) \subseteq I(A)$, $(t, (r, \mathrm{Im}(x))) \in I$. Therefore $t \in \mathrm{FinComp}(L_{A,A}) \subseteq \mathrm{FinComp}(L)$.

Conversely, let $t \in \mathrm{FinComp}(L)$, there exist some $u \in L$ and $v \in \mathbb{C}(X, D)$ such that $u = tv$ with $(t, v) \in I$. Let $B = \mathrm{alph}(t)$ and $A = I(B)$, note that $B \subseteq I(A)$. We claim that $t = \mu_A(u)$ and $u \in L_{A,A}$ whence $t \in \nu_A(L)$. Let $v = (r, D(C))$, since $(t, v) \in I$, we have by definition $B \times (\mathrm{alph}(r) \cup C) \subseteq I$, and thus $(\mathrm{alph}(r) \cup C) \subseteq I(B) = A$. Hence $\mathrm{alph}(r) \subseteq A$, in particular $\mathrm{alph}(r) \cap I(A) = \emptyset$ and then $\mu_A(u) = t$. We get also $C \subseteq A$, hence $u \in L_{A,A}$ and the claim is proved.

ii) Let NC be the set of non-connected complex traces. It is easy to verify that $NC = \bigcup_{(A \cup B) \times (E \cup F) \subseteq I} K(A, B) \cdot K(E, F)$ where, for all $A, B \subseteq X$, $K(A, B) = \{x \in \mathbb{C}(X, D)_B \setminus \{1\} \mid \mathrm{alph}(\mathrm{Re}(x)) = A\}$. Now, it turns out that

$$\mathrm{C}(L) = (\bigcup_{(A \cup B) \times (E \cup F) \subseteq I} (K(A, B)^{-1} \cdot L) \cap K(E, F)) \cap (\mathbb{C}(X, D) \setminus NC)$$

Clearly, for all $A, B \subseteq X$, the set $K(A, B)$ is recognizable. From Proposition 3.3 and Theorems 3.5 and 3.10, we deduce that NC and $\mathrm{C}(L)$ are recognizable. Thus the set $\mathrm{FC}(L) = \mathrm{C}(L) \cap \mathbb{M}(X, D)$ is recognizable, too.□

Theorem 4.3 *Let* $L \subseteq \mathbb{C}(X, D)$ *be recognizable. If* L *is finitary-connected (in particular if* L *is connected) or if* $\mathrm{FC}(L) \subseteq L^*$*, then* L^* *and* L^ω *are recognizable.*

Proof: If L is finitary connected, then $\mathrm{FinComp}(L)$ is connected and from Lemma 4.2 i), it holds that $\nu_A(L)$ is connected for all $A \subseteq X$.

Since by Lemma 3.4 $\nu_A(L)$ is also recognizable, $\nu_A(L)^*$ is recognizable by [Mét86, Och85]. Hence L^* and L^ω are recognizable by Corollary 3.9.

Assume that $\mathrm{FC}(L) \subseteq L^*$. We will also show that $\nu_A(L)^*$ is recognizable for all $A \subseteq X$. For this purpose, we claim that $\nu_A(L)^* = \nu_A(\mathrm{FC}(L))^*$.

Remark first that if t is a finite connected trace, from Lemma 4.2 i) applied to $L = \{t\}$, we get that, for any $A \subseteq X$, either $\nu_A(t) = \{t\}$ if $\mathrm{alph}(t) \subseteq I(A)$ or $\nu_A(t) = \emptyset$ otherwise.

Let now t be in $\nu_A(L)$ and $\{t_1, \ldots, t_k\}$ be the connected components of t. From Lemma 4.2 i), we have $\{t_1, \ldots, t_k\} \subseteq \mathrm{C}(\mathrm{FinComp}(L)) = \mathrm{FC}(L)$. From the remark above, for all $1 \leq i \leq k$, we get $\{t_i\} = \nu_A(t_i)$ and thus $t_i \in \nu_A(\mathrm{FC}(L))$. Hence $\nu_A(L) \subseteq \nu_A(\mathrm{FC}(L))^*$, and $\nu_A(L)^* \subseteq \nu_A(\mathrm{FC}(L))^*$, too.

Conversely, since $\mathrm{FC}(L) \subseteq L^*$, we have $\nu_A(\mathrm{FC}(L)) \subseteq \mathrm{FinComp}(\mathrm{FC}(L)) = \mathrm{FC}(L) \subseteq L^*$. Let $t \in \nu_A(\mathrm{FC}(L)) \subseteq L^*$, in particular $\mathrm{alph}(t) \subseteq I(A)$ and $t = t_1 \ldots t_k$ with $t_i \in L_\emptyset$ and $\mathrm{alph}(t_i) \subseteq I(A)$ for $1 \leq i \leq k$. From the remark above, $\{t_i\} = \nu_A(t_i) \subseteq \nu_A(L)$, for $1 \leq i \leq k$. Therefore $t \in \nu_A(L)^*$ and $\nu_A(\mathrm{FC}(L)) \subseteq \nu_A(L)^*$. Finally $\nu_A(L)^* = \nu_A(\mathrm{FC}(L))^*$ and the claim is proved.

Since $\nu_A(\mathrm{FC}(L))$ is connected, we obtain from [Mét86, Och85] that $\nu_A(L)^*$ is recognizable. □

Corollary 4.4 *Let $L \subseteq \mathbb{C}(X, D)$ be a recognizable, then $(\mathrm{FC}(L) \cup L)^*$ and $(\mathrm{FC}(L) \cup L)^\omega$ are recognizable, too.*

Proof: This is immediate from the theorem above and the observation that we have $\mathrm{FC}(\mathrm{FC}(L) \cup L) = \mathrm{FC}(L)$. □

Beside its own interest, Corollary 4.4 yields a simple proof of the following result which is surprising, since the $*$-iteration of a single trace is not recognizable, in general.

Corollary 4.5 *Let $x \in \mathbb{C}(X, D)$ be a recognizable complex trace, i.e., $\{x\} \in \mathrm{Rec}(\mathbb{C}(X, D))$, then x^ω is recognizable.*

Proof: Choose $A \subseteq X$ such that $D(A) = \mathrm{Im}(x^\omega)$. Then we claim that

$$\{x^\omega\} = ((\mathrm{FC}(\{x\}) \cup \{x\})^\omega)_A$$

One inclusion is obvious. Conversely, let $z \in ((\mathrm{FC}(\{x\}) \cup \{x\})^\omega)_A$. Since $\mathrm{Im}(z) = \mathrm{Im}(x^\omega) = D(\mathrm{alph}(\mathrm{Re}(x))) \cup \mathrm{Im}(x)$, we deduce that $z = zx = zx^\omega$. Now, let $\mathrm{FC}(\{x\}) = \{x_1, \ldots, x_k\}$ and let $y = (x_1 \ldots x_k)^{-1} \cdot x$. From the definition of connected components, it holds that $x_1, \ldots, x_k, y$ are pairwise independent. Therefore $z = x_1^{n+n_1} \ldots x_k^{n+n_k} y^n$ for some $0 \leq n_1, \ldots, n_k, n \leq \omega$, $x^\omega = x_1^\omega \ldots x_k^\omega y^\omega$ and hence $zx^\omega = x^\omega$. □

Now, we extend the concurrent iteration of Ochmanski [Och85] to complex traces.

Definition: *Let $L \subseteq \mathbb{C}(X, D)$. Then the concurrent iterations of L are defined by $L^{c-*} = (\mathrm{C}(L))^*$ and $L^{c-\omega} = (\mathrm{C}(L))^\omega$.*

Since $\mathrm{C}(L)$ is connected, we obtain from Lemma 4.2 and Theorem 4.3:

Corollary 4.6 *Let $L \subseteq \mathbb{C}(X, D)$ be recognizable. Then L^{c-*} and $L^{c-\omega}$ are recognizable.*

Definition: *The family of c-rational complex trace languages,*c $\Leftrightarrow$Rat($\mathbb{C}(X, D)$), *is the smallest family which contains all finite sets of finite traces and which is closed under the operations union, concatenation and concurrent iterations c-$*$ and c-ω.*

We are now ready to state our main result.

Theorem 4.7 $\mathrm{Rec}(\mathbb{C}(X, D)) =$ c $\Leftrightarrow$Rat$(\mathbb{C}(X, D))$

Proof: We have seen that $\mathrm{Rec}(\mathbb{C}(X, D))$ is a boolean algebra closed under concatenation (Theorem 3.5) and concurrent iterations (Corollary 4.6). Hence c $\Leftrightarrow$Rat$(\mathbb{C}(X, D)) \subseteq \mathrm{Rec}(\mathbb{C}(X, D))$. Conversely, we have seen (Corollary 3.6) that every recognizable language can be written as a finite union over languages of type $MN_1^\omega \ldots N_k^\omega s$ where $M, N_1, \ldots, N_k \subseteq \mathbb{M}(X, D)$ are recognizable and $s \in \mathbb{M}(X, D)$ is a finite trace. Moreover all

traces in N_i have the same connected alphabet and hence $N_i^\omega = N_i^{c-\omega}$. From Ochmanski's theorem [Och85], the recognizable finitary languages M and N_i, for $1 \leq i \leq k$, are in c $\Leftrightarrow$Rat($\mathbb{C}(X, D)$). Therefore L is also in c $\Leftrightarrow$Rat($\mathbb{C}(X, D)$). $\square$

Remark 4.8 i) Every recognizable complex trace language has effectively a c-rational expression where the concurrent iterations are allowed only for finitary languages which are connected. (Hence where the concurrent iterations become the usual Kleene-$*$ and ω-iteration.)

ii) We could have defined the finitary concurrent iteration of a language L as $L^{fc-\dagger} = (\mathrm{FC}(L) \cup L)^\dagger$ for $\dagger \in \{*, \omega\}$. An analogue of Theorem 4.7 with a suitable definition of fc $\Leftrightarrow$Rat($\mathbb{C}(X, D)$) also holds in this case.

5 Conclusion

In this paper we have studied the families of rational and recognizable languages of complex traces. We showed that classical results on finite and infinite words and finite traces have a natural extension to complex traces. In our investigations we followed mainly an algebraic viewpoint with the final goal to establish the Kleene-Ochmanki-Theorem.

Of course, other characterizations of the recognizable languages are of high interest. For instance, word languages can be defined by monadic second order logic [Büc62]. The equivalent work for finite traces can be found in [Tho90] and for for real traces in [Ebi92]. Its generalization to complex traces seems to be without any additional difficulty. Besides, the most well-known characterizations of recognizable languages use the notion of finite automata. Recognizable word languages are languages which are accepted by (non-deterministic) Büchi automata or by deterministic Muller automata. Asynchronous (cellular) automata [Zie87, Zie89] provide distributed acceptors for recognizable languages of finite traces. With a suitable Büchi like condition, non-deterministic

asynchronous (cellular) automata accept exactly the recognizable languages of real traces [GP92] and may be easily extended to accept complex trace languages. Hence, the main open problem in this field is to obtain a characterization of recognizable languages of infinite traces with an appropriate model of finite deterministic automata.

Acknowledgements: We thank Anca Muscholl for fruitful discussions and for the simple example before Theorem 2.14.

References

[AR88] I.J. Aalbersberg and G. Rozenberg. Theory of traces. *Theoret. Comput. Sci.*, 60:1–82, 1988.

[Arn85] A. Arnold. A syntactic congruence for rational ω-languages. *Theoret. Comput. Sci.*, 39:333–335, 1985.

[BMP90] P. Bonizzoni, G. Mauri, and G. Pighizzini. About infinite traces. In V. Diekert, editor, *Proceedings of the ASMICS workshop Free Partially Commutative Monoids, Kochel am See (Germany) 1989*, Report TUM-I9002, Technical University of Munich, pages 1–10, 1990.

[Büc62] R. Büchi. On a decision method in restricted second order arithmetic. In *Proc. Internat. Congress on Logic, Methodology and Philosophy*, pages 1–11. Standford University Press, 1962.

[CF69] P. Cartier and D. Foata. *Problèmes combinatoires de commutation et réarrangements.* Lecture Notes in Mathematics 85. Springer, Berlin-Heidelberg-New York, 1969.

[CP85] R. Cori and D. Perrin. Automates et commutations partielles. *R.A.I.R.O.-Informatique Théorique et Applications*, 19:21–32, 1985.

[DGP91] V. Diekert, P. Gastin, and A. Petit. Recognizable complex trace languages. In A. Tarlecki, editor, *Proceedings of the 16th Symposium on Mathematical Foundations of Computer*

Science (MFCS'91), Kazimierz Dolny (Poland) 1991, Lecture Notes in Computer Science 520, pages 131–140. Springer, Berlin-Heidelberg-New York, 1991.

[Die90] V. Diekert. *Combinatorics on Traces.* Lecture Notes in Computer Science 454. Springer, Berlin-Heidelberg-New York, 1990.

[Die91] V. Diekert. On the concatenation of infinite traces. In C. Choffrut et al., editors, *Proceedings of the 8th Annual Symposium on Theoretical Aspects of Computer Science (STACS'91), Hamburg (Germany) 1991*, Lecture Notes in Computer Science 480, pages 105–117. Springer, Berlin-Heidelberg-New York, 1991.

[Ebi92] W. Ebinger. On logical definability of infinite trace languages. In V. Diekert et al., editors, *Proceedings ASMICS Workshop Infinite Traces, Tübingen (Germany) 1992*, Bericht 4/92, Fakultät Informatik, Universität Stuttgart, 1992. This volume pages 106–122.

[Fli74] M. Fliess. Matrices de Hankel. *J. Math. Pures et Appl.*, 53:197–224, 1974.

[FR82] M.P. Flé and G. Roucairol. On the serializability of iterated transactions. In *Proceedings ACM SIGACT-SIGOPS, Symposium on Principles of Distr. Comp., Ottawa (Canada) 1982*, pages 194–200, 1982.

[Gas90] P. Gastin. Infinite traces. In I. Guessarian, editor, *Proceedings of the Spring School of Theoretical Computer Science on Semantics of Systems of Concurrent Processes*, Lecture Notes in Computer Science 469, pages 277–308. Springer, Berlin-Heidelberg-New York, 1990.

[Gas91] P. Gastin. Recognizable and rational trace languages of finite and infinite traces. In C. Choffrut et al., editors, *Proceedings*

of the 8th Annual Symposium on Theoretical Aspects of Computer Science (STACS'91), Hamburg (Germany))1991, Lecture Notes in Computer Science 480, pages 89–104. Springer, Berlin-Heidelberg-New York, 1991.

[GP92] P. Gastin and A. Petit. Asynchronous cellular automata for infinite traces. In W. Kuich, editor, *Proceedings of the 19th International Colloquium on Automata Languages and Programming (ICALP'92), Vienna (Austria) 1992*, Lecture Notes in Computer Science. Springer, Berlin-Heidelberg-New York, 1992. Also available as Tech. Rep. 91-68, LITP, Université Paris 6, France, 1991.

[GPZ91] P. Gastin, A. Petit, and W. Zielonka. A Kleene theorem for infinite trace languages. In J. Leach Albert et al., editors, *Proceedings of the 18th International Colloquium on Automata Languages and Programming (ICALP'91), Madrid (Spain) 1991*, Lecture Notes in Computer Science 510, pages 254–266. Springer, Berlin-Heidelberg-New York, 1991.

[GR91] P. Gastin and B. Rozoy. The poset of infinitary traces. Technical Report, LITP 91.24, Université de Paris 6, 1991. To appear in *Theoret. Comp. Sci.*

[Gra81] R. Graham. Rudiments of Ramsey theory. Regional Conference Series in Mathematics 45, 1981.

[Kwi90] M. Kwiatkowska. A metric for traces. *Imform. Proc. Letters*, 35:129–135, 1990.

[Maz77] A. Mazurkiewicz. Concurrent program schemes and their interpretations. DAIMI Rep. PB 78, Aarhus University, Aarhus, 1977.

[Maz87] A. Mazurkiewicz. Trace theory. In W. Brauer et al., editors, *Petri Nets, Applications and Relationship to other Models of Concurrency*, Lecture Notes in Computer Science 255, pages 279–324. Springer, Berlin-Heidelberg-New York, 1987.

[Mét86] Y. Métivier. Une condition suffisante de reconnaissabilité dans un monoïde partiellement commutatif. *R.A.I.R.O.-Informatique Théorique et Applications*, 20:121–127, 1986.

[MR91] Y. Métivier and B. Rozoy. On the star operation in free partially commutative monoids. *International Journal of Foundations of Computer Science*, 2:257–265, 1991.

[Och85] E. Ochmanski. Regular behaviour of concurrent systems. *Bull. of the European Association for Theoretical Computer Science (EATCS)*, 27:56–67, Oct 1985.

[Och90] E. Ochmanski. Notes on a star mystery. *Bull. of the European Association for Theoretical Computer Science (EATCS)*, 40:252–257, Feb 1990.

[Per89] D. Perrin. Partial commutations. In G. Ausiello et al., editors, *Proceedings of the 16th International Colloquium on Automata, Languages and Programming (ICALP'89), Stresa (Italy) 1989*, Lecture Notes in Computer Science 372, pages 637–651. Springer, Berlin-Heidelberg-New York, 1989.

[PP91] D. Perrin and J.E. Pin. Mots Infinis. Technical Report, LITP 91.06, Université de Paris 6, 1991.

[Sak87] J. Sakarovitch. On regular trace languages. *Theoret. Comput. Sci.*, 52:59–75, 1987.

[Tho90] W. Thomas. On logical definability of trace languages. In V. Diekert, editor, *Proceedings of the ASMICS workshop Free Partially Commutative Monoids, Kochel am See, Oktober 1989*, Report TUM-I9002, Technical University of Munich, pages 1–10, 1990.

[Zie87] W. Zielonka. Notes on finite asynchronous automata. *R.A.I.R.O.-Informatique Théorique et Applications*, 21:99–135, 1987.

[Zie89] W. Zielonka. Safe executions of recognizable trace languages by asynchronous automata. In A. R. Mayer et al., editors, *Proceedings Symposium on Logical Foundations of Computer Science, Logic at Botik '89, Pereslavl-Zalessky (USSR) 1989*, Lecture Notes in Computer Science 363, pages 278–289. Springer, Berlin-Heidelberg-New York, 1989.

Open problems in trace theory proposed by Giovanni Pighizzini and Nicoletta Sabadini

1 Determinization of asynchronous automata

A very surprising result obtained by W. Zielonka [Zie87] is the solution of the following problem:

Problem 1 *Given a deterministic automaton over the free partially commutative monoid find a deterministic asynchronous automaton accepting the same trace language.*

The proof given by Zielonka is rather complicated and it seems difficult to simplify it, despite all efforts, e.g. [CM88].

The characterization of the class of recognizable trace languages given by Ochmański [Och85] suggests a different approch to Problem 1 similar to the usual proof of Kleene's Theorem [Pig92]. Unfortunately, the automaton so obtained is nondeterministic. Even if the possibility of obtaining an equivalent deterministic asynchronous automaton is a consequence of Zielonka's result, up to date there is not any direct construction.

Thus, we state the following problem:

Open problem 1 *Find a "new" strategy to transform nondeterministic asynchronous automata in deterministic asynchronous automata, without using Zielonka's proof.*

We briefly point out that the main difficult of Zielonka's result is to obtain a distributed automaton starting from a monoid automaton, in such a way that there is a link between their states.

Our open problem seems to be easier since the input is already a distributed automaton; in spite of this, similar technical problems arise.

Similar considerations can be formulated for asynchronous cellular automata.

2 Probabilistic asynchronous automata

As a second open problem, we still propose the generalization of Zielonka's Theorem to probabilistic asynchronous automata.

Probabilistic asynchronous automata were introduced in [MS89]. In [JPS90], the following problem were positively solved in the case of concurrent alphabets with acyclic dependency graph.

Open problem 2 *Given a concurrent alphabet* (Σ, C)*, does the class of behaviours of* $F(\Sigma, C)$*–probabilistic asynchronous automata coincide with the class of behaviours of probabilistic asynchronous automata over* (Σ, C) *?*

In the general case this problem is still open.

3 Extensions of the notion of infinite trace

An extension of the notion of finite traces was recently proposed by A. Arnold [Arn91], by considering pomsets without self–concurrency. In this extension the concurrency relation is not fixed *a priori*. A similar extension in the infinite case could be useful to describe infinite behaviours of concurrent systems. So, we state the following problem:

Open problem 3 *Extend the theory of infinite traces to pomsets without self–concurrency, giving a suitable notion of recognizing devices.*

4 The equivalence and the inclusion problems for k–ambiguous regular trace languages

The *equivalence problem (inclusion problem,* resp.) for a class $\mathcal{C}$ of trace languages consists in deciding if $L' = L''$ ($L' \subseteq L''$, resp.), where L' and L'' are languages in the class $\mathcal{C}$.

Let $R_k(\Sigma, C)$ denotes the class of regular trace languages over the concurrent alphabet (Σ, C) with degree of ambiguity k [BMS82b] and $R_0(\Sigma, C)$ the class of recognizable trace languages. We briefly recall that

$$R_0(\Sigma, C) \subseteq R_1(\Sigma, C) \subseteq \ldots \subseteq R_k(\Sigma, C) \subseteq \ldots \subseteq R(\Sigma, C),$$

where $R(\Sigma, C)$ is the class of regular trace languages over (Σ, C).

It is known that for the class $R_0(\Sigma, C)$ the equivalence problem is decidable for all concurrent alphabets, while for the class $R(\Sigma, C)$ it is decidable if and only if the concurrency relation C is transitive [BMS82a, AH87]. In [BPS88], it was proved that there exists a concurrent alphabet (Σ, C) with nontransitive concurrency relation such that the equivalence problem for $R_1(\Sigma, C)$ is decidable. This result has been recently extended in [Var91] proving that for every concurrent alphabet (Σ, C), the equivalence problem for $R_1(\Sigma, C)$ is decidable.

Up to date, there is none result concerning the classes $R_k(\Sigma, C)$ for $k \geq 2$.

Open problem 4 *For every $k \geq 2$, characterize the class of concurrent alphabets (Σ, C) such that the equivalence problem for trace languages in $R_k(\Sigma, C)$ is decidable.*

For the class $R(\Sigma, C)$ of regular trace languages, the inclusion problem is decidable if and only if the independency relation C is transitive [Sak90]. Moreover, the same problem for $R_0(\Sigma, C)$ is decidable for every concurrent alphabet.

None result is known for the class $R_1(\Sigma, C)$ of unambiguous regular trace languages and for k–ambiguous regular trace languages.

Open problem 5 *For every $k \geq 1$, characterize the class of concurrent alphabets (Σ, C) such that the inclusion problem for trace languages in $R_k(\Sigma, C)$ is decidable.*

References

[AH87] IJ. Aalbersberg and H. Hoogeboom. Decision problems for regular trace languages. In *Proc. 14th ICALP*, Lecture Notes in Computer Science 267, pages 251–259, 1987.

[Arn91] A. Arnold. An extension of the notions of traces and of asynchronous automata. *RAIRO Inf. Theor.*, 25:355–393, 1991.

[BMS82a] A. Bertoni, G. Mauri, and N. Sabadini. Equivalence and membership problems for regular trace languages. In *Proc. 9th ICALP*, Lecture Notes in Computer Science 140, pages 61–71, 1982.

[BMS82b] A. Bertoni, G. Mauri, and N. Sabadini. A hierarchy of regular trace languages and some combinatorial applications. In *Proc. 2nd World Conference on Mathematics at the Service of Men*, pages 146–153, Las Palmas, 1982.

[BPS88] D. Bruschi, G. Pighizzini, and N. Sabadini. On the existence of the minimum asynchronous automaton and on decision problems for unambiguous regular trace languages. In *Proc. 5th STACS*, Lecture Notes in Computer Science 294, pages 334–346, 1988. To appear in *Information and Computation*.

[CM88] R. Cori and Y. Métivier. Approximation of a trace, asynchronous automata and the ordering of events in a distributed system. In *Proc. 15th ICALP*, Lecture Notes in Computer Science 317, pages 147–161, 1988.

[JPS90] S. Jesi, G. Pighizzini, and N. Sabadini. Probabilistic asynchronous automata. In *Proceedings of the workshop: Free partially commutative monoids*, Institüt für Informatic – Technische Universität München – TUM–I9002, pages 99–114, 1990.

[MS89] P. Massazza and N. Sabadini. Some applications and techniques for generating functions. In *Proc. CAAP 89*, Lecture Notes in Computer Science 351, pages 321–336, 1989.

[Och85] E. Ochmański. Regular behaviour of concurrent systems. *EATCS Bullettin*, 27:56–67, 1985.

[Pig92] G. Pighizzini. Synthesis of nondeterministic asynchronous automata. 1992. These proceedings.

[Sak90] J. Sakarovitch. The "last" decision problem on rational trace languages. In *Proceedings of the workshop: Free partially commutative monoids*, Institüt für Informatic – Technische Universität München – TUM–I9002, pages 168–171, 1990.

[Var91] S. Varricchio. On the decidability of the equivalence problem for partially commutative rational power series, 1991. Manuscript.

[Zie87] W. Zielonka. Notes on finite asynchronous automata. *RAIRO Inf. Theor.*, 21:99–135, 1987.