Trace Theory

Volker Diekert Anca Muscholl

April 2011

Abstract

Trace Theory denotes a mathematical theory of free partially commutative monoids from the perspective of concurrent or parallel systems. Traces, or equivalently, elements in a free partially commutative monoid, are given by a sequence of letters (or atomic actions). Two sequences are assumed to be equal if they can be transformed into each other by equations of type $ab = ba$, where the pair (a, b) belongs to a predefined relation between letters. This relation is usually called *partial commutation* or *independence*. With an empty independence relation, i.e., without independence, the setting coincides with the classical theory of words or strings.

Introduction

The analysis of sequential programs describes a run of a program as a sequence of atomic actions. On an abstract level such a sequence is simply a string in a free monoid over some (finite) alphabet of letters. This purely abstract viewpoint embeds program analysis into a rich theory of combinatorics on words and a theory of automata and formal languages. The approach has been very fruitful from the early days where the first compilers have been written until now where research groups in academia and industry develop formal methods for verification.

Efficient compilers use autoparallelization which provides a natural example of independence of actions resulting in a partial commutation relation. For example, let $a; b; c; a; d; e; f$ be a sequence of arithmetic operations where:

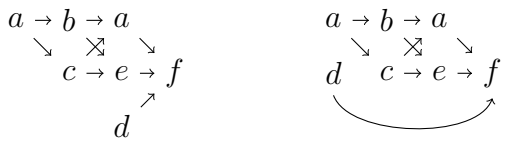
$$\begin{array}{lll} (a)\ x := x + 2y, & (b)\ x := x - z, & (c)\ y := y \cdot 5z \\ (d)\ w := 2w, & (e)\ z := y \cdot z, & (f)\ z := x + y \cdot w. \end{array}$$

A concurrent-read-exclusive-write protocol yields a list of pairs of independent operations (a, d), (a, e), (b, c), (b, d), (c, d), and (d, e), which can

be performed concurrently or in any order. The sequence can therefore be performed in four parallel steps $\{a\}\,;\{b,c\}\,;\{a,d,e\}\,;\{f\}$, but as d commutes with a, b, c the result of $a; b; c; a; d; e; f$ is equal to $a; d; b; c; a; e; f$, and two processors are actually enough to guarantee minimal parallel execution time, since another possible schedule is $\{a,d\}\,;\{b,c\}\,;\{a,e\}\,;\{f\}$. Trace theory yields a tool to do such (data-independent) transformations automatically.

Parallelism and concurrency demand for specific models, because a purely sequential description is neither accurate nor possible in all cases, for example if asynchronous algorithms are studied and implemented. Several formalisms have been proposed in this context. Among these models there are Petri nets, Hoare's CSP and Milner's CCS, event structures, and branching temporal logics. The mathematical analysis of Petri nets is however quite complicated and much of the success of Hoare's and Milner's calculus is due to the fact that is stays close to the traditional concept of sequential systems relying on a unified and classical theory of words. Trace theory follows the same paradigm, it enriches the theory of words by a very restricted, but essential formalism to capture main aspects of parallelism: In a static way a set I of independent letters (a, b) is fixed, and sequences are identified if they can be transformed into each other by using equations of type $ab = ba$ for $(a, b) \in I$. In computer science this approach appeared for the first time in the paper by Keller on *Parallel Program Schemata and Maximal Parallelism* published in 1973. Based on the ideas of Keller and the behavior of elementary net systems Mazurkiewicz introduced 1977 the notion of *trace theory* and made its concept popular to a wider computer science community. Mazurkiewicz' approach relies on a graphical representation for a trace. This is a node-labeled directed acyclic graph, where arcs are defined by the dependence relation, which is by definition the complement of the independence relation I.

Thereby, a concurrent run has an immediate graphical visualization which is obviously convenient for practice. The picture of the two parallel executions $\{a\}\,;\{b,c\}\,;\{a,d,e\}\,;\{f\}$ and $\{a,d\}\,;\{b,c\}\,;\{a,e\}\,;\{f\}$ can be depicted as follows, which represents (the Hasse diagrams of) isomorphic labeled partial orders:



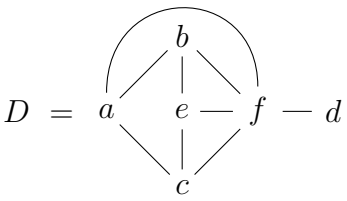
Moreover, the graphical representation yields immediately a correct notion of *infinite trace*, which is not clear when working with partial commutations. In the following years it became evident that trace theory indeed

copes with some important phenomena such as *true concurrency*. On the other hand it is still close to the classical theory of word languages describing sequential programs. In particular, it is possible to transfer the notion of finite sequential state control to the notion of asynchronous state control. This important result is due to Zielonka; it is one of the highlights of the theory. There is a satisfactory theory of recognizable languages relating finite monoids, rational operations, asynchronous automata, and logic. This leads to decidability results and various effective operations. Moreover it is possible to develop a theory of asynchronous Büchi automata, which enables in trace theory the classical automata theory based approach to automated verification.

1 Discussion

1.1 Mathematical definitions and normal forms

Trace theory is founded on a rigorous mathematical approach. The underlying combinatorics for partial commutation were studied in mathematics already in 1969 in the seminal Lecture Notes in Mathematics *Problèmes combinatoires de commutation et réarrangements* by Cartier and Foata. The mathematical setting uses a finite alphabet Σ of letters and the specification of a symmetric and irreflexive relation $I \subseteq \Sigma \times \Sigma$, called the *independence* relation. Conveniently, its complement $D = \Sigma \times \Sigma \subseteq I$ is called the *dependence* relation. The dependence relation has a direct interpretation as graph as well. For the dependency used in the first example above it looks as follows:



The intended semantics is that independent letters commute, but dependent letters must be ordered. Taking $ab = ba$ with $(a, b) \in I$ as defining relations one obtains a quotient monoid $\mathbb{M}(\Sigma, I)$, which has been called *free partial commutative monoid* or simply *trace monoid* in the literature. The elements are finite (Mazurkiewicz-)traces. For $I = \emptyset$, traces are just words in Σ^*; for a full independence relation, i.e., $D = \mathrm{id}_\Sigma$, traces are vectors in some $\mathbb{N}^k$, hence Parikh-images of words. The general philosophy is that the

extrema Σ^* and $\mathbb{N}^k$ are well-understood (which is far from being true), but the the interesting and difficult problems arise when $\mathbb{M}(\Sigma, I)$ is neither free nor commutative.

For effective computations and the design of algorithms appropriate normal forms can be used. For the *lexicographic* normal form it is assumed that the alphabet Σ is totally ordered, say $a < b < c \cdots < z$. This defines a lexicographic ordering on Σ^* exactly the same way words are ordered in a standard dictionary. The lexicographic normal form of a trace is the minimal word in Σ^* representing it. For example, if I is given by $\{(a,d),(d,a),(b,c),(c,a)\}$, then the trace defined by the sequence *badacb* is the congruence class of six words:

$$\{baadbc, badabc, bdaabc, baadcb, badacb, bdaacb\}\,.$$

Its lexicographic normal form is the first word *baadbc*. An important property of lexicographic normal forms has been stated by Anisimov and Knuth. A word is in lexicographic normal form if and only if it does not contain a *forbidden pattern*, which is a factor *bua* where $a < b \in \Sigma$ and the letter a commutes with all letters appearing in $bu \in \Sigma^*$. As a consequence, the set of lexicographic normal forms is a regular language.

The other main normal is due to Foata. It is a normal form which encodes a maximal parallel execution. Its definition uses *steps*, where a step means here a subset $F \subseteq \Sigma$ of pairwise independent letters. Thus, a step requires only one parallel execution step. A step F yields a trace by taking the product $\Pi_{a \in F} a$ over all its letters in any order. The *Foata normal form* is a sequence of steps $F_1 \cdots F_k$ such that $F_1, \ldots, F_k$ are chosen from left-to-right with maximal cardinality. The sequence $\{a,d\}\,;\{b,c\}\,;\{a,e\}\,;\{f\}$ above has been the Foata normal form of $abcadef$.

The graphical representation of a trace due to Mazurkiewicz can be viewed as a third normal form. It it called the *dependence graph representation*; and it is closely related to the Foata normal form. Say a trace t is specified by some sequence of letters $t = a_1 \cdots a_n$. Each index $i \in V = \{1, \ldots, n\}$ is labeled by the letter a_i. Finally, arcs $(i,j) \in E$ are introduced if and only if both $(a_i, a_j) \in D$ and $i < j$. In this way an acyclic directed graph $G(t)$ is defined which is another unique representation of t. The information about t is also contained in the induced partial order (i.e., the transitive closure of $G(t)$) or in its Hasse-diagram (i.e., removing all transitive arcs from $G(t)$).

1.1.1 Computation of normal forms

There are efficient algorithms which compute either normal form in polynomial time. A very simple method uses a stack for each letter of the alphabet Σ. An input word is scanned from right to left, so the last letter is read first.

When processing a letter a it is pushed on its stack and a marker is pushed on the stack of all the letters b ($b \neq a$) which do not commute with a. Once the word has been processed its lexicographic normal form, the Foata normal form, and the Hasse-diagram of the dependence graph representation can be obtained straightforwardly. For example, the sequence $a; b; c; a; d; e; f$ (with a dependence relation as depicted above) yields stacks as follows:

					$*$
					$*$
a	$*$	$*$			$*$
$*$	b	c		$*$	$*$
$*$	$*$	$*$		$*$	$*$
a	$*$	$*$	d	e	$*$
$*$	$*$	$*$	$*$	$*$	f
a	b	c	d	e	f

1.2 Regular sets

A fundamental concept in formal languages is the notion of a *regular set*. Kleene's Theorem say that a regular set can be specified either by a finite deterministic (resp. non-deterministic) automaton DFA (resp. NFA) or, equivalently, by a regular expression. Regular expressions are also called *rational expressions*. They are defined inductively by saying that every finite set denotes a rational expression and if R, S rational, then $R \cup S$, $R \cdot S$, and R^* are rational expressions, too. The semantics of a rational expression is defined in any monoid M since the semantics of $R \cup S$, $R \cdot S$ is obvious, and R^* can be viewed as the union $\bigcup_{k \in \mathbb{N}} R^k$. For *star-free* expressions one does not allow the star-operation, but one adds complementation, denoted e.g. by $\overline{R}$ with the semantics $M \setminus R$.

In trace theory a direct translation of Kleene's Theorem fails, but it can be replaced by a generalization due to Ochmański. If (a, b) is a pair of independent letters, then $(ab)^*$ is a rational expression, but due to $ab = ba$ it represents all strings with an equal number of a's and b's which is clearly not regular. With three pairwise independent letters $(abc)^*$ is not even context-free. A general formal language theory distinguishes between recognizable and rational sets. A subset L of a trace monoid is called *recognizable*, if its closure is a regular word language. Here the closure refers to all words in Σ^* which represent some trace in L. A subset L is called *rational*, if L can be specified by some regular (and hence rational) expression. Using the algebraic notion of homomorphism this can be rephrased as follows. Let φ be the canonical homomorphism of Σ^* onto $\mathbb{M}(\Sigma, I)$, which simply means the interpretation of a string as its trace. Now, L is recognizable if and only if

$\varphi^{-1}(L)$ is a regular word language; and L is rational if and only if $L = \varphi(K)$ for some regular word language K. As a consequence of Kleene's Theorem all recognizable trace languages are rational, but the converse fails as soon as there is a pair of independent letters, i.e., the trace monoid is not free.

Given a recognizable trace language L, the corresponding word language $\varphi^{-1}(L)$ is accepted by some NFA (actually some DFA) which satisfies the so-called *I-diamond property*. This means whenever it holds $(a, b) \in I$ and a state p leads to a state q by reading the word ab, then it is in state p also possible to read ba and this leads to state q, too. NFAs satisfying the I-diamond property accept closed languages only. Therefore they capture exactly the notion of recognizability for traces.

It has been shown that the concatenation of two recognizable trace languages is recognizable, in particular *star-free languages* (i.e., given by star-free expressions) are recognizable. However, the example $(ab)^*$ above shows that the star-operation leads to non-recognizable sets as soon the trace monoid is not free. Métivier and Ochmański have introduced a restricted version where the star-operation is allowed only when applied to languages L where all traces $t \in L$ are connected. This means the dependence graph $G(t)$ is connected or, equivalently, there is no non-trivial factorization $t = uv$ where all letters in u are independent of all letters in v. A theorem shows that L^* is still recognizable, if L is connected (i.e., all $t \in L$ are connected) and recognizable. Ochmański's Theorem yields also the converse: A trace language L is recognizable if and only if it can be specified by a rational expression where the star-operation is restricted to connected subsets. As word languages are always connected this is a proper generalization of the classical Kleene's Theorem. Yet another characterization of recognizable trace languages is as follows: They are in one-to-one correspondence with regular subsets inside the regular set LexNF $\subseteq \Sigma^*$ of lexicographic normal forms. The correspondence associates with $L \subseteq \mathbb{M}(\Sigma, I)$ the set $K = \varphi^{-1}(L) \cap \text{LexNF}$. A rational expression for K is a rational expression for L, where the star-operation is restricted to connected languages.

1.3 Decidability questions

1.3.1 The Star Mystery

The *Star Problem* is to decide for a given recognizable trace language $L \subseteq \mathbb{M}(\Sigma, I)$ whether L^* is recognizable. It is not known whether the star problem is decidable, even if it is restricted to finite languages L. The surprising difficulty of this problem has been coined as the *star mystery* by Ochmański. It has been shown by Richomme that the Star Problem is decidable, if (Σ, I)

does not contain any C_4 (cycle of four letters) as an induced subgraph.

1.3.2 Undecidability results for rational sets

For rational languages (unlike as for recognizable languages) some very basic problems are known to be undecidable. The following list contains undecidable decision problems, where the input for each instance consists of an independence alphabet (Σ, I) and rational trace languages $R, T \subseteq \mathbb{M}(\Sigma, I)$ specified by rational expressions.

- **Inclusion** Question: Does $R \subseteq T$ hold?
- **Equality** Question: Does $R = T$ hold?
- **Universality** Question: Does $R = \mathbb{M}(\Sigma, I)$ hold?
- **Complementation** Question: Is $\mathbb{M}(\Sigma, I) \setminus R$ a rational?
- **Recognizability** Question: Is R recognizable?
- **Intersection** Question: Does $R \cap T = \emptyset$ hold?

On the positive side, if I is transitive, then all six problems above are decidable. This is also a necessary condition for the first five problems in the list. Transitivity of the independence alphabet alphabet means in algebraic terms that the trace monoid is a free product of free and free commutative monoids, like e.g. $\{a, b\}^* * \mathbb{N}^3$.

The intersection problem is simpler. It is known that the problem Intersection is decidable if and only if (Σ, I) is a transitive forest. It is also well-known that transitive forests are characterized by forbidden induced subgraphs C_4 and P_4 (cycle and path, resp., of four letters).

1.4 Asynchronous automata

Whereas recognizable trace languages can be defined as word languages accepted by DFAs or NFAs with I-diamond property, there is an equivalent distributed automaton model called *asynchronous automata*. Such an automaton is a parallel composition of finite-state processes synchronizing over shared variables, whereas a DFA satisfying the I-diamond property is still a device with a centralized control. An asynchronous automaton $\mathcal{A}$ has, by definition, a distributed finite state control such that independent actions may be performed in parallel. The set of global states is modeled as a direct product $Q = \prod_{p \in P} Q_p$, where the Q_p are states of the local component $p \in P$

and P is some finite index set (a set of processors). For each letter $a \in \Sigma$ there is a *read domain* $R(a) \subseteq P$ and a *write domain* $W(a) \subseteq P$ where for simplicity $W(a) \subseteq R(a)$. Processors p and q share a variable a if and only if $p, q \in R(a)$. The transitions are given by a family of partially defined functions δ_p, where each processor p reads the status in the local components of its read domain and changes states in local components of its write domain. Accordingly to the read-and-write-conflicts being allowed, four basic types are distinguished:

- Concurrent-Read-Exclusive-Write (*CREW*),
  if $R(a) \cap W(b) = \emptyset$ for all $(a, b) \in I$.
- Concurrent-Read-Owner-Write (*CROW*),
  if $R(a) \cap W(b) = \emptyset$ for all $(a, b) \in I$ and $W(a) \cap W(b) = \emptyset$ for all $a \neq b$.
- Exclusive-Read-Exclusive-Write (*EREW*),
  if $R(a) \cap R(b) = \emptyset$ for all $(a, b) \in I$.
- Exclusive-Read-Owner-Write (*EROW*),
  if $R(a) \cap R(b) = \emptyset$ for all $(a, b) \in I$ and $W(a) \cap W(b) = \emptyset$ for all $a \neq b$.

The local transition functions $(\delta_p)_{p \in P}$ give rise to a partially defined transition function on global states $\delta : (\prod_{p \in P} Q_p) \times \Sigma \longrightarrow \prod_{p \in P} Q_p$.

If $\mathcal{A}$ is of any of the four types above, then the action of a trace $t \in \mathbb{M}(\Sigma, I)$ on global states is well-defined. This allows to see an asynchronous automaton as an I-diamond DFA. There are effective translations from one model to the other. The most compact versions can be obtained by a CREW model, therefore it is of prior practical interest.

Zielonka has shown in his thesis (published in 1987) the following deep theorem in trace theory: Every recognizable trace language can be accepted by some finite asynchronous automaton. The proof of this theorem is very technical and complicated. Moreover, the original construction was doubly exponential in the size of an I-diamond automaton for the language L. Therefore it is part of ongoing research to simplify its construction, in particular since efficient constructions are necessary to make the result applicable in practice. The best result to date is due to Genest et al.. They provide a construction where the size of the obtained asynchronous automaton is polynomial in the size of a given DFA and simply exponential in the number of processes. They also show that the construction is optimal within the class of automata produced by Zielonka-type constructions, which yields a non-trivial lower bound on the size of asynchronous automata.

A rather direct construction of asynchronous automata is known for triangulated dependence alphabets, this means that all chordless cycles are of length three. For example, complete graphs and forests are triangulated.

1.5 Infinite traces

The theory of infinite traces has its origins in the mid eighties when Flé and Roucairol considered the problem of serializability of iterated transactions in data bases. A suitable definition of an infinite trace uses the dependence graph representation due to Mazurkiewicz. Just as in the finite case an infinite sequence $t = a_1 a_2 \cdots$ of letters yields an infinite node-labeled acyclic directed graph $G(t)$, where now each $i \in V = \mathbb{N}$ is labeled by the letter a_i, and again arcs $(i, j) \in E$ are introduced if and only if both $(a_i, a_j) \in D$ and $i < j$. It is useful to consider finite and infinite objects simultaneously as an infinite trace may split into connected components where some of them might be finite. The notion of *real trace* has been introduced to denote either a finite or an infinite trace. If $t_1, t_2, \ldots$ is (finite or infinite) sequence of finite traces, then the product $t_1 t_2 \cdots$ is a well-defined real trace. It is a finite trace if almost all t_i are empty and an infinite trace otherwise. In particular, one can define the ω-product L^ω for every set L of finite traces and one enriches the set of rational expressions by this operation.

The set $\mathbb{R}(\Sigma, I)$ of real traces can be embedded into a monoid of *complex traces* where the *imaginary* component is a subset of Σ. This alphabetic information is necessary in order to define an associative operation of concatenation. (Over complex traces L^ω is defined for all subsets L.)

Many results from the theory of finite traces transfer to infinite traces according to the same scheme as for finite and infinite words.

1.6 Logics

1.6.1 MSO and first-oder logic

Formulae in monadic second-order logic (MSO) are built up upon first-order variables $x, y, \ldots$ ranging over vertices and second-order variables $X, Y, \ldots$ ranging over subsets of vertices. There are Boolean constants *true* and *false*, the logical connectives $\vee$, $\wedge$, $\neg$, and quantification $\exists$, $\forall$ for the first- and second-order variables. In addition there are four types of atomic formulae:

$$x \in X,\ x = y,\ (x, y) \in E, \text{ and } \lambda(x) = a \ .$$

A first-order formula is a formula without any second-order variable. A *sentence* is a closed formula, i.e., a formula without free variables. The semantics

of an MSO-sentence is defined for every node-labeled graph $[V, E, \lambda]$ (here: V = set of vertices, E = set of edges, $\lambda : V \to \Sigma$ = vertex labeling). Identifying a trace t with its dependence graph $G(t)$, the truth value of $t \models \psi$ is therefore well-defined for every sentence ψ. The trace language defined by a sentence ψ is $L(\psi) = \{t \in \mathbb{R}(\Sigma, I) \mid t \models \psi\}$. It follows a notion of first-order and second-order definability of trace languages.

1.6.2 Temporal logic

Linear temporal logic LTL can be inductively defined inside first order as formulae with one free variable, as soon as the transitive closure $(x, y) \in E^*$ is expressible in first-order (as it is the case for trace monoids). There are no quantifiers, but all Boolean connectives. The atomic formulae are $\lambda(x) = a$. If $\varphi(x), \psi(x)$ are LTL-formulae, then $\mathsf{EX}\,\varphi(x)$ and $(\varphi\,\mathsf{U}\,\psi)(x)$ are LTL-formulae. In temporal logic $(x, y) \in E^*$ means that y is in the future of the node x. The semantics of $\mathsf{EX}\,\varphi(x)$ is *exists next*, thus $\varphi(y)$ holds for a direct successor of x. The semantics of $(\varphi\,\mathsf{U}\,\psi)(x)$ reflects an *until operator*, it says that in the future of x there is some z which satisfies $\psi(z)$ and all y in the future of x but in the strict past of z satisfy $\varphi(y)$. Hence, condition φ holds until ψ becomes true. There are dual past-tense operators, but they do not add expressivity.

For LTL on can also give a syntax without any free variable and a *global semantics* where the evaluation is based on the prefix relation of traces. The local semantics as defined above is for traces a priori expressively weaker, but it was shown that both, the global and local LTL have the same expressive power as first-order logic. This was done by Thiagarajan and Walukiewicz in 1998 for global LTL and by Diekert and Gastin in 2006 for local LTL, respectively. Both results extend a famous result of Kamp from words to traces. The complexity of the satisfiability problem (or model checking) is however quite different. In global semantics it is non-elementary, whereas in local semantics it is in PSPACE (= class of problems solvable on a Turing machine in polynomial space.)

1.7 Fragments

For various applications fragments of first-order logics suffice. This has the advantage that simpler constructions are possible and that the complexity of for model checking is possibly reduced. A prominent fragment is first-order logic with at most two names for variables. Two-variable logics logics capture the core features of XML navigational languages like XPath. Over words and over traces two variable logic $\mathrm{FO}^2[E]$ can be characterized algebraically via

the variety of monoids DA, (referring to the fact that regular $\mathcal{D}$- classes are aperiodic semigroups), in logic by *Next-Future* and *Yesterday-Past* operators, and in terms of rational expressions via unambiguous polynomials. It turns out that the satisfiability problem for two-variable logic is NP-complete (if the independence alphabet is not part of the input). The extension of these results from words to traces is due to Kufleitner.

1.8 Logics, algebra, and automata

The connection between logic and recognizability uses algebraic tools from the theory of finite monoids. If $h : \mathbb{M}(\Sigma, I) \to M$ is a homomorphism to a finite monoid M and $L \subseteq \mathbb{R}(\Sigma, I)$ is a set of real traces, then one says that h recognizes L, if for all $t \in L$ and factorizations $t = t_1 t_2 \cdots$ into finite traces t_i the following inclusion holds: $h^{-1}(t_1)h^{-1}(t_2)\cdots \subseteq L$. This allows to speak of *aperiodic languages* if some recognizing monoid is aperiodic. A monoid M is *aperiodic*, if for all $x \in M$ there is some $n \in \mathbb{N}$ such that $x^{n+1} = x^n$. A deep result states that a language is first-order definable if and only if it is recognized by a homomorphism to a finite aperiodic monoid. Algebraic characterizations lead to decidability of fragments. For example it is decidable whether a recognizable language is aperiodic or whether it can be expressed in two-variable first-order logic.

Another way to define recognizability is via Büchi automata. A Büchi automaton for real traces is an I-diamond NFA with a set of final states F and a set of repeated states R. It accepts a trace if the run stops in F or if repeated states are visited infinitely often. If its transformation monoid is aperiodic it is called aperiodic, too. There is also a notion of asynchronous (cellular) Büchi automaton, and it is known that every I-diamond Büchi automaton can be transformed into an equivalent asynchronous cellular Büchi automaton.

The main result connecting logic, recognizability, rational expressions, and algebra can be summarized by saying that the following statements in the first block (second block resp.) are equivalent for all trace languages $L \subseteq \mathbb{R}(\Sigma, I)$:

MSO definability:

1 L is definable in monadic second-order logic.

2 L is recognizable by some finite monoid.

3 L is given as a rational expression where the star is restricted to connected languages.

4 L is accepted by some asynchronous Büchi automaton.

First-order definability:

1 L is definable in first-order logic.

2 L is definable in LTL (with global or local semantics).

3 L is recognizable by some finite and aperiodic monoid.

4 L is star-free.

1.8.1 Automata based verification

The automata theoretical approach to verification uses the fact that systems and specifications are both modeled with finite automata. More precisely, a system is given as a finite transition system $\mathcal{A}$ which is typically realized as an NFA without final states. So, the system allows finite and infinite runs. The specification is written in some logical formalism, say in the linear temporal logic LTL. So the specification is given by some formula φ, and its semantics $L(\varphi)$ defines the runs which obey the specification. Model checking means to verify the inclusion $L(\mathcal{A}) \subseteq L(\varphi)$. This is equivalent to $L(\mathcal{A}) \cap L(\neg\varphi) = \emptyset$. Once an automaton $\mathcal{B}$ with $L(\mathcal{B}) = L(\neg\varphi)$ has been constructed, standard methods yield a product automaton for $L(\mathcal{A}) \cap L(\mathcal{B})$. The check for emptiness becomes a reachability problem in directed graphs.

A main obstacle is the combinatorial explosion when constructing the automaton $\mathcal{B}$. But this works in practice nevertheless reasonable well, because typical specifications are simple enough to be understood (hopefully) by the designer, so they are short. From a theoretical viewpoint the complexity of model checking for MSO and first-order in non-elementary, but for (local) LTL is still in PSPACE. This approach is mostly applied and very successful where runs can be modeled as sequences. Trace theory provides the necessary tools to extend these methods to asynchronous systems. A first step in this direction has been implemented in the framework of *partial order* reduction. Another application of trace theory is the analysis of communication protocols.

1.9 Traces and asynchronous communication

Trace automata like asynchronous ones model concurrency in the same spirit as Petri nets, using shared variables. A more complex model arises when concurrent processes cooperate over unbounded, fifo communication channels.

A *communicating automaton* is defined over a set P of processes, together with point-to-point communication channels $Ch \subseteq \{(p,q) \in P^2 \mid p \neq q\}$. It consists of a tuple of NFAs $\mathcal{A}_p$, one for each process $p \in P$. Each NFA $\mathcal{A}_p$ has a set of local states Q_p and transition relation $\delta_p \subseteq Q_p \times \Sigma_p \times Q_p$. The set Σ_p of local actions of process p consists of send-actions $p!q(m)$ (of message m to process q, $(p,q) \in Ch$) and receive-actions $p?r(m)$ (of message m from process r, $(r,p) \in Ch$), respectively. The semantics of such an automaton is defined through configurations consisting of a tuple of local states (one for each process) and a tuple of word contents (one for each channel). In terms of partial orders the semantics of runs corresponds to *message sequence charts (MSC)*, a graphical notation for fifo message exchange. In contrast with asynchronous automata, communicating automata have an infinite state space and are actually Turing powerful, thus most algorithmic questions about them are undecidable.

The theory of recognizable trace languages enjoys various nice results known from word languages, e.g. in terms of logics and automata. Since communicating automata are Turing powerful, one needs restrictions in order to obtain e.g. logical characterizations. A natural restriction consists in imposing bounds on the size of the channels. Such bounds come in two versions, namely as *universal* and *existential* bounds, respectively. The existential version of channel bounds is optimistic and considers all those runs that can be rescheduled on bounded channels. The universal version is pessimistic and considers only those runs that, independent of the scheduling, can be executed with bounded channels. Thus, communicating automata with an universal channel bound are finite state, whereas with an existential channel bound they are infinite state systems.

Kuske proposed an encoding of runs of communicating automata with bounded channels into trace languages. Using this encoding, the set of runs (MSCs) of a communicating automaton is the projection of a recognizable trace language (for a universal bound), respectively the set of MSCs generated by the projection of a recognizable trace language (for an existential bound). This correspondence has the same flavor as the distinction between recognizable and rational trace languages, respectively.

The logic MSO over MSCs is defined with an additional binary message-predicate relating matching send and receive events. Henriksen et al. and Genest et al., respectively, have shown that the equivalence between MSO and automata extends to communicating automata with universal and existential channel bound, respectively. Another equivalent characterization exists in terms of MSC-graphs, similar to star-connected expressions for trace languages. These expressiveness results are complemented by decidable instances of the model-checking problem.

2 Bibliographic notes and further reading

Trace theory has its origins in enumerative combinatorics when Cartier and Foata found a new proof of the MacMahon Master Theorem in the framework of partial commutation by combining algebraic and bijective ideas [2]. The Foata normal form was defined in this Lecture Note. In computer science the key idea to use partial commutation as tool to investigate parallel systems was laid by Keller [10], but it was only by the influence of the technical report of Mazurkiewicz [11] when these ideas were spread to a wider computer science community, in particular to the Petri-net community. It was also Mazurkiewicz who coined the notion *Trace theory* and who introduced the notion of dependence graphs as a visualization of traces. The characterization of lexicographic normal forms by forbidden pattern is due to Anisimov and Knuth [1].

The investigation of recognizable (regular, rational resp.) languages is central in the theory of traces. The characterization of recognizable languages in terms of star-connected regular expressions is due to Ochmański [13]. The notion of *asynchronous automaton* is due to Zielonka. The major theorem showing that all recognizable languages can be accepted by asynchronous automata is his work (built on his thesis) [15]. The research on asynchronous automata is still an important and active area. The best constructions so far are due to Genest et al., where also non-trivial lower bounds were established [8].

The theory of infinite traces has its origins in the mid eighties. A definition of a real trace as a prefix-closed and directed subset of real traces and its characterization by dependence graphs is given in a survey by Mazurkiewicz [12]. The theory of recognizable real trace languages has been initiated by Gastin in 1990. The generalization of the Kleene-Büchi-Ochmański-Theorem to real traces is due to Gastin, Petit and Zielonka [7]. Diekert and Muscholl gave a construction for deterministic asynchronous Muller automata accepting a given recognizable real trace language.

Ebinger initiated the study of LTL for traces in his thesis in 1994. But it took quite an effort until Diekert and Gastin were able to show that LTL (in local semantics) has the same expressive power as first-oder logic [3]. The advantage of a local LTL is that model checking in PSPACE, whereas in its global semantics it becomes non-elementary by a result of Walukiewicz [14]. The PSPACE-containment has been shown for a much wider class of logics by Gastin and Kuske [6]. In [4] Diekert, Horsch, and Kufleitner give a survey on fragments of first order logic in trace theory. The Büchi-like equivalence between automata and MSO for existentially bounded communicating automata has been shown by Genest, Kuske and Muscholl in [9].

The translation from MSO into automata uses the equivalence for trace languages, but needs some additional, quite technical construction specific to communicating automata.

Very much of the material used in the present discussion can be found in The Book of Traces which was edited by Diekert and Rozenberg [5]. The book surveys also a notion of *semi-commutation* (introduced by Clerbout and Latteux), and it provides many hints for further reading. Current research efforts concentrate on the topic of distributed games and controller synthesis for asynchronous automata.

Volker Diekert, FMI, Universität Stuttgart, Germany
email: diekert@fmi.uni-stuttgart.de

Anca Muscholl, LaBRI, Université Bordeaux 1, France
email: anca@labri.fr

References

[1] Anatolij V. Anisimov and Donald E. Knuth. Inhomogeneous sorting. *International Journal of Computer and Information Sciences*, 8:255–260, 1979.

[2] Pierre Cartier and Dominique Foata. *Problèmes combinatoires de commutation et réarrangements*. Number 85 in Lecture Notes in Mathematics. Springer-Verlag, Heidelberg, 1969.

[3] Volker Diekert and Paul Gastin. Pure future local temporal logics are expressively complete for Mazurkiewicz traces. *Information and Computation*, 204:1597–1619, 2006. Conference version in LATIN 2004, LNCS 2976, 170–182, 2004.

[4] Volker Diekert, Martin Horsch, and Manfred Kufleitner. On first-order fragments for Mazurkiewicz traces. *Fundamenta Informaticae*, 80:1–29, 2007.

[5] Volker Diekert and Grzegorz Rozenberg, editors. *The Book of Traces*. World Scientific, Singapore, 1995.

[6] Paul Gastin and Dietrich Kuske. Uniform satisfiability in PSPACE for local temporal logics over Mazurkiewicz traces. *Fundam. Inform.*, 80(1-3):169–197, 2007.

[7] Paul Gastin, Antoine Petit, and Wiesław Zielonka. An extension of Kleene's and Ochmański's theorems to infinite traces. *Theoretical Computer Science*, 125:167–204, 1994.

[8] Blaise Genest, Hugo Gimbert, Anca Muscholl, and Igor Walukiewicz. Optimal Zielonka-type construction of deterministic asynchronous automata. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *ICALP (2)*, volume 6199 of *Lecture Notes in Computer Science*, pages 52–63. Springer, 2010.

[9] Blaise Genest, Dietrich Kuske, and Anca Muscholl. A Kleene theorem and model checking algorithms for existentially bounded communicating automata. *Inf. Comput.*, 204(6):920–956, 2006.

[10] Robert M. Keller. Parallel program schemata and maximal parallelism I. Fundamental results. *Journal of the Association for Computing Machinery*, 20(3):514–537, 1973.

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

[12] Antoni Mazurkiewicz. Trace theory. In W. Brauer et al., editors, *Petri Nets, Applications and Relationship to other Models of Concurrency*, number 255 in Lecture Notes in Computer Science, pages 279–324, Heidelberg, 1987. Springer-Verlag.

[13] Edward Ochmański. Regular behaviour of concurrent systems. *Bulletin of the European Association for Theoretical Computer Science (EATCS)*, 27:56–67, October 1985.

[14] Igor Walukiewicz. Difficult configurations – on the complexity of LTrL. In Kim G. Larsen et al., editors, *Proc. 25th International Colloquium Automata, Languages and Programming (ICALP'98), Aalborg (Denmark)*, number 1443 in Lecture Notes in Computer Science, pages 140–151, Heidelberg, 1998. Springer-Verlag.

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