# Introduction

To understand the behavior of contemporary computing devices, the concept of parallelism or concurrency is inevitable. There are several obvious reasons for an increasing use of these techniques: In an attempt to make programs faster one may distribute them over several executing machines. By duplicating memory or computation tasks, the reliability of systems can be increased. On a certain level of abstraction, a specification is inherently concurrent since the subsystems are thought to run independently from each other. Another aspect is that communication networks consist by definition of independent subsystems that are only loosely coupled. These observations call for a deeper understanding of the mechanisms involved.

For sequential systems, a mathematical foundation has proved fruitful. Already the consideration of formal systems in the first half of this century laid the ground for a distinction between (theoretically) computable and definitely not mechanizable tasks. Complexity theory sharpens this distinction further by the investigation of the frontier between tractable and nontractable computational tasks. Finite automata, although they are a very restricted model of a sequential computing device, have a rich theory as well as a wide-spread application. Their theory is closely related to algebraic theories. Furthermore, surprising connections between different logics and automata were found. These connections make it possible to automatize certain verification tasks in the development of software and hardware systems.

Aiming at similar benefits, attempts to develop a mathematical formalization of parallelism have a longstanding tradition in computer science. In the 60s, Petri introduced nets, now called Petri nets, as a model of concurrent systems. These Petri nets inspired many theoretical investigations and now have an extensive theory. But the semantics of these nets is technically quite complicated and a mathematical treatment in its full generality turns out to be cumbersome. Another line of research in this area is based on the idea of a process algebra introduced by Milner and Hoare in the 70s. This approach focuses more on programming languages. Cellular automata can be traced back to ideas of v. Neumann but became widely known only in the 70s (in particular by Conway's "Game of Life"). Now they enjoy a well understood theory as well as several extensions.

Mazurkiewicz introduced traces, another model of concurrent behaviors, into computer science. They can be defined in two equivalent ways, either as dependence graphs or as equivalence classes of words. In both cases, one starts out from a finite set of elementary or atomic actions, called alphabet, and a binary dependence relation on the set of actions. Two such actions are dependent if they e.g. use a common resource. Hence, in a parallel computation of the system, independent actions can be performed in parallel, while dependent actions can be performed sequentially, only. A computation of such a system is modeled as a directed graph. The vertices of this graph correspond to events. Two such vertices are connected by an edge iff their labels are dependent. Since the computation is meant to run in time, the graph is assumed to be acyclic. Furthermore, we consider only finite computations and therefore finite graphs. A dependence graph is nothing but such a finite directed acyclic graph with edges between dependent vertices. Thus, a dependence graph describes the causal dependence in a computation.

In the alternative definition, one considers sequential observations of some parallel computation. The order in which independent actions are performed is regarded as irrelevant. In particular, if two observations differ only in the order of independent actions, they are identified. This defines an equivalence relation on words (over the alphabet of actions) and a trace is an equivalence class with respect to this equivalence.

It turns out that the linear extensions of a dependence graph form an equivalence class, i.e. a trace, and that any trace can be obtained from some dependence graph. In this sense, the two approaches are equivalent and "it is only a matter of taste which objects are chosen for representing concurrent processes: equivalence classes of strings or labelled graphs" [Maz95, page 14]. It seems that this dual nature of traces has contributed to a large extent to their success. This is not the right place to recall the vast amount of results on traces that have been obtained in the last two decades. For in-depth surveys on the results on traces, the reader is referred to [DR95] that concentrates on the theoretical aspects in computer science as well as in mathematics, in particular in combinatorics.

Nonetheless, it turned out that certain limitations of traces made it necessary to extend the model into different directions. The probably most modest extension was that to infinite or real traces. These were introduced to model not only finite but also infinite computations. They can be defined in several equivalent ways: as directed and downward closed sets of finite traces [Maz87], via an equivalence relation on infinite words [Sta89, Kwi90] or as infinite dependence graphs where any event dominates only a finite number of vertices. Diekert introduced $\alpha$- and $\delta$-complex traces as metric completion of the set of finite traces with respect to two different metrics [Die91, Die93] and showed in particular that they can alternatively be defined as infinite dependence graphs with some alphabetic information. Most of the considerations on complex traces are based on this second characterization. Another similar extension of traces (approximating traces) is presented in the same spirit [DG98].

The generalizations mentioned so far have been introduced to model infinite behaviors of a parallel system. Differently, the aim of semi-commutations is to model some behaviors like the producer-consumer-example that do not fit into the setting of a symmetric independence relation. The idea is that the consumption of an item can be delayed after further productions, but conversely, the production cannot be postponed after the consumption. Here, we refer the reader to the survey [CLR95] and the literature cited therein.

Another limitation of Mazurkiewicz traces is the global and fixed independence relation. There are certainly systems where the answer to the question whether two particular actions can be performed concurrently depends on the situation, e.g. on the available resources that are produced by preceding events (cf. [KP92]). An automaton with concurrency relations [Dro90, Dro92] is a (finite) automaton whose states are equipped with independence relations, i.e. in this model the dependence of actions can change while the system evolves. Similarly to traces, one obtains an equivalence relation on the set of finite computation sequences by identifying those sequences that differ only in the order of independent actions. But now this independence refers to the state were the first of the two actions is performed. Thus, originally the behavior of an automaton with concurrency relations was defined via equivalence classes of sequential behaviors. In [BDK95, BDK97], representing these computations by dependence graphs, we presented a partial order semantics for these computations under some mild assumptions on the automaton with concurrency relations.

Another approach to incorporate changing independence relations into the model of traces is represented by context and generalized traces [BR94]. Here, two actions might be independent in one context and dependent in another where the context is (in the simplest form) given by the preceding action. Again, first an equivalence of words was constructed and context traces were defined as equivalence classes of words. An attempt to represent context traces by dependence graphs succeeded only partially [BR95].

Common to all generalizations listed so far is that the independence of actions is a binary relation. This limits their applicability since it is not possible to model a situation were two items of some resource are claimed by three actions. In such a situation, any two of the claiming actions might be performed concurrently and the third one afterwards. In addition, traces and their successors do not allow autoconcurrency. Local traces [HKT92, Hoo94] are an attempt to solve these problems. Here, sets or even multisets of actions are declared independent and this depends on the history of the system. A representation of such systems by local event structures was obtained in the same papers. In [KM00], we give a representation of computations in this model by dependence graphs.

Note that in all the extensions mentioned so far, computations were first modeled as equivalence classes of sequential executions. Later (for some models much later) it was shown that these equivalence classes can be nicely represented by structures like dependence graphs. Differently, P-traces are by definition labeled

partially ordered sets. Afterwards it is shown that they can also be obtained as equivalence classes of certain equivalence relations [Arn91].

Besides this duality, the different extensions of Mazurkiewicz traces have been considered under several aspects. Mazurkiewicz used traces to model the behavior of one-safe Petri nets. Categorical adjunctions were constructed between larger classes of Petri nets and trace structures [NRT90], step transition systems (i.e. local traces) [Muk92] and concurrent automata [DS93]. The order theoretic properties of the set of all trace-like objects was investigated for real traces [GR93, BCS93, Kus99], for several versions of complex traces [GP92, Teo93, DG98], and for the computations of an automaton with concurrency relations [Sta89, Dro90, Dro92, Kus94a, Kus94b, Sch98]. Metric and topological questions were dealt with for real traces [Kwi90, KK00], for complex and approximating traces [Die91, Die93, DG98], and for computations of automata with concurrency relations [KS98]. The recognizable sets of trace-like structures were studied thoroughly. The relation to rational sets was investigated for semi-commutations, for real and for complex traces (cf. the corresponding surveys in [DR95]), and for computations of concurrent automata [Dro94, Dro95, Dro96]. The relation to logically axiomatizable sets can be found for finite and for real traces in [Tho90b, EM93, Ebi94], for computations of concurrent automata in [DK96, DK98] and for local traces in [KM00].

In the first part of the current work, we will define an extension of dependence graphs to so called $\Sigma$-dags where $\Sigma$ is a finite set of actions. They generalize not only dependence graphs as defined above, but also CCI-sets [Arn91], dependence graphs of computations of concurrent automata [BDK95, BDK97], and (width-bounded) sp-pomsets [LW98b, LW98a, LW00]. Essentially, a $\Sigma$-dag is a $\Sigma$-labeled directed acyclic graph. The edges of this graph represent the causal dependency between the events that are modeled by the vertices. There are only two restrictions that we impose: First, we allow no autoconcurrency. Second, for any label $a$, an event can depend on and influence at most one $a$-labeled event directly.

As a computational model for these $\Sigma$-dags, we investigate asynchronous cellular automata. They were defined originally for dependence graphs as a truly parallel accepting device [Zie87].[1] Since then, they have been intensively studied, cf. [Zie95, DM95] for overviews. In [DG96], they were generalized in such a way that an asynchronous cellular automaton can accept labeled posets (pomsets) without autoconcurrency (cf. also [Kus98, DGK00]). Here, we extend them to the setting of $\Sigma$-dags. In the literature, infinite state systems are intensively studied [Mol96, BE97]. We extend asynchronous cellular automata furthermore by allowing them to have infinitely many states. To preserve some finiteness, the set of states is endowed with a well-quasi ordering. Thus, loosely speaking, asynchronous cellular machines or $\Sigma$-ACMs are asynchronous cellular automata

[1] The name might be misleading since these automata are not a generalization of v. Neumann's cellular automata mentioned above.

that run on $\Sigma$-dags, have possibly infinitely many states, and are equipped with a well-quasi ordering on these states.

The behavior of a $\Sigma$-ACM is the accepted language, i.e. a set of $\Sigma$-dags. Hence a $\Sigma$-ACM describes a property of $\Sigma$-dags. Since the intersection as well as the union of two acceptable sets can be accepted by a $\Sigma$-ACM, properties describable by $\Sigma$-ACMs can become quite complex. Then it is of interest whether the combined property is contradictory, or, equivalently, whether at least one $\Sigma$-dag satisfies it. Thus, one would like to know whether a $\Sigma$-ACM accepts at least one $\Sigma$-dag. Using a result by Finkel & Schnoebelen [FS98a, FS98b] on well-structured transition systems, we show that it is possible to gain this knowledge even automatically, i.e. we show that there exists an algorithm that on input of a $\Sigma$-ACM decides whether the $\Sigma$-ACM accepts at least one $\Sigma$-dag. For this to hold, we restrict the asynchronous cellular machines in two ways: The notion of "monotonicity" involves a connection between the well-quasi ordering and the transitions of the machine. The notion "effectiveness" requires that the machine is given in a certain finite way.

Another natural question is whether two properties are equivalent, i.e. whether two $\Sigma$-ACMs accept the same language. Since there is a $\Sigma$-ACM that accepts all $\Sigma$-dags, a special case of this equivalence problem is to ask whether a given $\Sigma$-ACM accepts all $\Sigma$-dags. The latter question, called universality, essentially asks whether the described property is always true. The corresponding question for sequential automata has a positive answer which is a consequence of the decidability of the emptiness: If one wants to know whether a sequential automaton accepts all words, one constructs the complementary automaton and checks whether its languages is empty. Thus, the crucial point for sequential automata is that they can effectively be complemented. But the set of acceptable $\Sigma$-dag-languages is not closed under complementation. Therefore, we cannot proceed as for sequential automata. On the contrary, we show that the universality is undecidable even for $\Sigma$-ACMs with only finitely many states. These finite $\Sigma$-ACMs are called asynchronous cellular automata or $\Sigma$-ACA. The undecidability of the universality implies that the equivalence of two $\Sigma$-ACAs, the complementability of a $\Sigma$-ACA as well as the existence of an equivalent deterministic $\Sigma$-ACA are undecidable, too. These undecidability results (restricted to pomsets) together with a sketch of proof were announced in [Kus98]. The proof we give here is based on ideas developed together with Paul Gastin.

The following chapter deals with the question which properties can be expressed by a $\Sigma$-ACA. For finite sequential automata, several answers are known to the question which properties can be checked by a finite sequential automaton: Kleene showed that these are precisely the rational properties. By the Myhill-Nerode Theorem, a property can be checked by a finite sequential automaton if its syntactic monoid is finite. Furthermore, Büchi and Elgot [Büc60, Elg61] showed that a property of words can be checked by a finite sequential automaton if it can be expressed in the monadic second order logic. This relation between a model

of a computational device (finite sequential automata) and monadic second order logic is a paradigmatic result. It has been extended in several directions, e.g. to infinite words [Büc60], to trees [Rab69] (cf. also [Tho90a]), to finite [Tho90b] and to real [EM93, Ebi94] traces, and to computations of concurrent automata [DK96, DK98]. The celebrated theorem of Zielonka [Zie87, Zie95] together with the results from [Tho90b] states that for dependence graphs of traces, the expressive power of asynchronous cellular automata and monadic second order logic coincide. Aiming at a similar result for $\Sigma$-dags, in Chapter 5 we show that this is not possible in general. More precisely, we show that any recognizable set of $\Sigma$-dags can be axiomatized by a sentence of the monadic second order logic, but that the converse is false even for first-order logic. To overcome this, we restrict to a subclass of all $\Sigma$-dags, called $(\Sigma, k)$-dags. This restriction makes it possible to relabel a $(\Sigma, k)$-dag by an asynchronous cellular automaton in such a way that one obtains a dependence graph over a certain dependence alphabet. This is the crucial step in our proof that any monadically axiomatizable set of $(\Sigma, k)$-dags can be accepted by a (nondeterministic) asynchronous cellular automaton. But we show that it is necessary to allow nondeterminism in the automata since the expressive power of deterministic $\Sigma$-ACAs will be proved to be strictly weaker. Again, the restriction to pomsets of the results presented in this chapter can be found in [Kus98]. Here, we generalize the presentation in [DGK00].

The final chapter of the first part is devoted to the relation between our asynchronous cellular automata and other models of concurrent behavior. The covering relation of a pomset without autoconcurrency is a $\Sigma$-dag. This allows us to speak of the set of pomsets that is accepted by a $\Sigma$-ACA: A pomset $(V, \leq, \lambda)$ is accepted iff its Hasse-diagram $(V, \prec, \lambda)$ admits a successful run. For pomsets, other automata models have been proposed in the literature. In particular, Arnold considered P-asynchronous automata [Arn91] and Lodaya & Weil dealt with branching automata [LW98a, LW98b, LW00]. We finish our consideration of $\Sigma$-dags and $\Sigma$-ACAs by a comparison of the expressive power of these automata with the expressive power of our $\Sigma$-ACAs. We show that branching automata when restricted to width-bounded languages have the same expressive power as monadic second order logic. This enables us to prove that $\Sigma$-ACAs and branching automata have the same expressive power. Finally, we show that any P-asynchronous automaton can be simulated by a $\Sigma$-ACA.

The $\Sigma$-dags considered in the first part of the current work are clearly labeled graphs. Above, I already cited A. Mazurkiewicz stating "it is only a matter of taste which objects are chosen for representing concurrent processes: equivalence classes of strings or labelled graphs." [Maz95, page 14]. To satisfy those that prefer the algebraic approach (or at least appreciate it as the author), this is followed in the second part where left divisibility monoids are considered. These left divisibility monoids were introduced in joint work with Manfred Droste [DK99, DK00]. As pointed out earlier, trace monoids are defined via a finite presentation (us-

ing a set of letters $\Sigma$ together with a dependence relation on $\Sigma$). Later, algebraic properties where discovered that characterize trace monoids (up to isomorphism) [Dub86]. Differently, left divisibility monoids are defined in the language of monoids, i.e. via their algebraic properties. In particular, it is required that the prefix relation be a partial order and that for any monoid element, the set of prefixes forms a distributive lattice. Thus, divisibility monoids involve monoid theoretic as well as order theoretic concepts.

In Chapter 8, we show that divisibility monoids can be finitely presented. Not only will we show that this is possible in general, but we will give a concrete representation for any divisibility monoid. Finally, we give a decidable class of finite presentations that give rise to all divisibility monoids.

Kleene's theorem on recognizable languages of finite words has been generalized in several directions, e.g. to formal power series [Sch61], to infinite words [Büc60], and to infinite trees [Rab69]. More recently, rational monoids were investigated [Sak87], in which the recognizable languages coincide with the rational ones. Building on results from [CP85, CM88, Mét86], a complete characterization of the recognizable languages in a trace monoid by c-rational sets was obtained in [Och85]. A further generalization of Kleene's and Ochmański's results to concurrency monoids was given in [Dro95]. In Chapter 9, we derive such a result for divisibility monoids. The proofs by Ochmański [Och85] and by Droste [Dro95] rely on the *internal* structure of the elements of the monoids. Here, we do not use the internal representation of the monoid elements, but algebraic properties of the monoid itself. Thus, the considerations in Chapter 9 that appeared in [DK99] can be seen as an algebraic proof of Ochmański's Theorem.

The following chapter is devoted to the question when a divisibility monoid satisfies Kleene's Theorem, i.e. when the rational and the recognizable sets coincide. For trace monoids, this is only the case if the trace monoid is free. Our result states that a divisibility monoid satisfies Kleene's Theorem iff it is a rational monoid [Sak87]. A defining property of divisibility monoids is that the sets of prefixes form a distributive lattice for any element of the monoid. We prove that this set of distributive lattices is width-bounded iff the monoid satisfies Kleene's Theorem. We obtain these characterizations applying the theory of rational functions (cf. [Ber79]) and a Foata normal form of monoid elements similar to that for traces.

Büchi showed that the monadic second order theory of the linearly ordered set $(\omega, \leq)$ is decidable. To achieve this goal, he used finite automata. In the course of these considerations he showed that a language in a free finitely generated monoid is recognizable iff it is monadically axiomatizable. In computer science, this latter result and its extension to infinite words are often referred to as "Büchi's Theorem" while in logic this term denotes the decidability of the monadic theory of $\omega$. In the final chapter, I understand it in this second meaning. There, we show that certain monadic theories associated to a divisibility monoid are decidable. Let $\mathfrak{L}$ denote the set of distributive lattices associated to a given divisibility

monoid. We show that the monadic theory of this class is decidable iff the monoid satisfies Kleene's Theorem. In general, this theory is undecidable, but the monadic theory of the join-irreducible elements of these lattices is still decidable. For trace monoids, this latter result just states that the monadic theory of all dependence graphs is decidable, a corollary from [EM93, Ebi94].

At the very end, we prove an order theoretic result that is inspired by the two decidabilities just mentioned: Together with a result from Chapter 10, we know that the monadic theory of $\mathfrak{L}$ is decidable if and only if $\mathfrak{L}$ is width-bounded. In a certain sense, we show that this does not depend on the special character of $\mathfrak{L}$ as the set of lattices associated with a divisibility monoid. Indeed, we show that any set of finite distributive lattices $\mathfrak{L}$ has a decidable monadic theory if and only if the monadic theory of the join-irreducible elements of these lattices is decidable and $\mathfrak{L}$ is width-bounded.

The present work shows that there are deep connections that arise from the theory of traces to different branches of mathematics. We finish the work with a list of problems that show up in the course of our considerations.