Computers have become an integral part of human lives and made it possible to solve many previously unattainable problems. Some tasks entrusted to them are very responsible. In situations when human life is at stake (e.g., when a computer is steering an autonomous car or conducting a medical surgery) there is no place for programming errors. There arises a need for dependable software systems which are provably error-free. This is one of the issues addressed by theoretical computer science, which developed various approaches to solving this problem.

One of them, verification, aims at proving that programs satisfy some given specification (describing their desired or undesired behaviours or properties). This is obtained through a rigorous analysis of programs, which often incorporates representing inner workings of real systems with simpler mathematical models. Another technique, known as program synthesis, assumes a different initial setup, where only specification needs to be provided, and the task is to automatically construct a program implementing it.

Both programs and their specifications can be represented through the means of models of computation. The trade-off between the expressive power of a model (which phenomena can they describe) and the simplicity of its analysis (ease of verifying its properties) cuts across the field of computation theory. A multitude of models exists, each resolving this difficult compromise in a different way.

On one side of the spectrum, there are finite automata—one of the simplest and most natural models. They are essentially finite transition systems labeled with letters from some alphabet—finite directed graphs with letter-labeled edges. Their computation, starting and ending in designated initial and final locations, corresponds to a string of letters visited in-between. Languages thereof form a class of regular languages—type three in the landmark Chomsky hierarchy. A well-studied\(^1\) class of models, many of their properties being easily verifiable, finite automata are easy to work with, but not very expressive.

\(^1\)Although here, too, there are few problems waiting to be solved, such as the question of size of the smallest automaton that distinguishes two given words of length \(n\).
1. Models of computation

On the opposite side lie Turing machines. Like finite automata, they are based on finite transition systems, but also feature a kind of memory: an infinite tape with read and write access. The machine starts with an input word written on the tape, and performs computations until reaching an accepting state. If there is no finite accepting computation, the word is rejected. This more complicated model is more powerful, as it can recognize exactly all recursively enumerable languages, which constitute level zero of the Chomsky hierarchy. Unfortunately, it is much harder to verify properties of Turing machines. Most natural decision problems, like question whether given machine ever stops, are undecidable—there does not exist a reliable algorithmic way of determining the answer.

There is a great number of models of computation situated between these extreme cases, some of them addressing specific needs of particular domain and differing in the trade-offs made between expressive power and simplicity of analysis. Compared to Turing machines, such in-between devices may operate on other types of memory, be subject to syntactic or semantic constraints, and vary in design, e.g., working on infinite alphabets. Custom models often offer better fit to the use-case for which they were created, while avoiding increased computational complexity or undecidability of problems.

It should come as no surprise that researchers often try to determine circumstances in which it is possible to effectively replace a complex model with a simpler device. The quest of finding less involved devices that approximate the behavior of a complex system with sufficient precision is the central theme of this dissertation. We focus on several instances of that general scheme, considering the following questions:

Q1 Given a complex specification, does there exist a simpler device realizing it?
    (synthesis problem)

Q2 Can disjoint languages of complex models be distinguished (separated) in a simpler way?
    (separability problem)

Q3 Can a complex device be equivalently described in a simpler way?
    (membership problem)

Q4 Is a complex system bound to contain a simpler, more structured subsystem?
    (pumping lemma)

Before we can state these problems more precisely and present our contributions in §2, we need to introduce the models of computation that are of interest in this dissertation.

1 Models of computation

AUTOMATA OVER INFINITE ALPHABETS

In the upcoming §§1.1 and 1.2, we present two models which both recognize words over infinite alphabets: $\Sigma \times \mathbb{Q}_{\geq 0}$ and $\Sigma \times \mathbb{A}$, where each input letter from a finite set $\Sigma$ is paired with an element of an infinite set. In case of the first model—timed automata—the element is a nonnegative rational number interpreted as a timestamp, while for the next—register automata—$\mathbb{A}$ is an arbitrary countably infinite set of atoms. Words over
these alphabets are called *timed words* and *data words*, and sets thereof are called timed and data languages, respectively. Both models are built upon standard finite automata. To deal with the extended alphabets, they are equipped with so-called *clocks* and *registers*, respectively.

1.1 Automata with clocks (NTA/DTA)

*Timed automata* are one of the most widespread models of real-time reactive systems. They are an extension of finite automata with rational-valued clocks\(^2\) which can be reset and compared by inequality constraints (which may use integer constants).

**Example 1.1.**

Timed words \(w_1 = (a, 0)(a, 4.5)\) and \(w_2 = (a, 0)(a, 5.7)\) can be distinguished by a timed automaton. Consider the following automaton \(A\) with one clock \(x\) and three states.

```
reset clock x if x < 5
```

It resets a clock \(x\) at the first symbol and then accepts if \(x < 5\) upon arrival of the next timed symbol. Note that the clocks measure time at the same pace as the timestamps of the input letters increase.

As with most models of computation, we distinguish between *nondeterministic* (NTA) and *deterministic* (DTA) timed automata.

**Nondeterministic timed automata**

The nonemptiness problem for *NTA* is decidable and, in fact, \(\text{PSPACE}\)-complete, as shown by Alur and Dill in their landmark 1994 paper [2]. This paved the way for the automatic verification of timed systems, leading to mature tools such as UPPAAL [8], UPPAAL Tiga (timed games) [14], and PRISM (probabilistic timed automata) [57]. The reachability problem is still a very active research area these days [1, 37, 40, 41, 44, 46], as are expressive generalisations thereof, such as the binary reachability problem [24, 34, 39, 56]. As a testimony to the model’s importance, the authors of [2] received the 2016 Church Award [16] for the invention of timed automata.

**Deterministic timed automata**

These are a strict subclass of *NTA*, where the successive configuration is uniquely determined by the current one and the timed input symbol. The class of *DTA* enjoys stronger properties than *NTA*, such as decidable universality/equivalence/inclusion problems, and closure under complementation [2]. Moreover, the more restrictive nature of *DTA* is needed for several applications of timed automata, such as test generation [65], fault diagnosis [10], learning [76, 81], winning conditions in timed games [4, 11, 49], and in a notion of recognisability of timed languages [60].

For these reasons, and for the more general quest of understanding the nature of the expressive power of nondeterminism in timed automata, many researchers have focused

---

\(^2\)This generally accepted name is not very accurate, because the clocks of timed automata have more affinity to *stopwatches* than to clocks, which cannot be easily reset.
on defining determinisable classes of timed automata, such as strongly non-zeno NTA [5], event-clock NTA [3], and NTA with integer-resets [74]. These classes are not exhaustive, in the sense that there are NTA recognising deterministic timed languages not falling into any of them. In fact, the class of determinisable NTA is undecidable. Thus, it is of interest to be able to decide whether a timed language presented as an NTA is actually recognisable by a DTA.

Another remarkable subclass of NTA is obtained by requiring the presence of just one clock (without epsilon transitions). The resulting class of NTA₁ is incomparable with DTA: For instance, NTA₁ are not closed under complement (unlike DTA), and there are very simple DTA languages that are not recognisable by any NTA₁. Nonetheless, NTA₁, like DTA, have decidable inclusion, equivalence, and universality problems [58,66], although the complexity is non-primitive recursive [58, Corollary 4.2] (see also [67, Theorem 7.2] for an analogous lower bound for the satisfiability problem of metric temporal logic). Moreover, the nonemptiness problem for NTA₁ is NLogSpace-complete (vs. PSPACE-complete for unrestricted NTA and DTA, already with two clocks [37]), and the binary reachability relation of NTA₁ can be computed as a formula of existential linear arithmetic of polynomial size, which is not the case in general [20].

1.2 Automata with registers (NRA/DRA)

The theory of register automata shares many similarities with that of timed automata. Nondeterministic register automata (NRA) have been introduced by Kaminski and Francez around the same time as timed automata [50]. They were defined as an extension of finite automata with finitely many registers which can store input values (now called data values, or atoms) and be compared with equality and disequality constraints.

Example 1.2. register automaton distinguishing two data words

Data words $w_1 = (a,0)(a,0)$ and $w_2 = (a,0)(a,1)$ can be distinguished by a register automaton $A$ with one register $x$ and three states:

![Diagram of register automaton]

Above, in the operations on edges, $y$ denotes the currently read input value, while 0, 1 stand for some particular atoms.

Kaminski and Francez have shown, among other things, that nonemptiness is decidable [50, Theorem 1]. It was later realised that the problem is, in fact, PSPACE-complete [33, Theorems 4.3 and Theorem 5.1]. The class of NRA recognisable languages is not closed under complementation [50, Proposition 5]; moreover, universality (and thus equivalence and inclusion) of NRA is undecidable [64, Theorem 5.1] (already for NRA with two registers [33, Theorem 5.4]).
Deterministic register automata

One way to regain decidability is to consider the deterministic register automata (DRA). This strict subclass of \( \text{nra} \) is effectively closed under complement and thus has decidable inclusion problem\(^3\) (entailing also universality and equivalence). DRA also provide the foundations of learning algorithms for data languages \([63]\). A recent result completing the theory of register automata has shown that a data language is DRA recognisable if, and only if, both this language and its complement are \( \text{nra} \) recognisable \([53]\).

As in the case of timed automata, it has been observed that restricting the number of registers results in algorithmic gains. Already in the seminal work of Kaminski and Francez, it has been proved that the inclusion problem \( \mathcal{L}(A) \subseteq \mathcal{L}(B) \) is decidable when \( A \) is an \( \text{nra} \) and \( B \) is an \( \text{nra} \) with one register \([50\text{, Appendix A}]\), albeit the complexity is non-primitive recursive in this case \([33\text{, Theorem 5.2}]\).

Other atoms

The notion of register automaton by Kaminski and Francez was subsequently generalised to allow input values (now commonly referred to as atoms) to originate from relational structure \( A \) given as a parameter. In this setting, the original variant of register automata is obtained by choosing \( A \) to be the equality atoms \( \langle \mathbb{N}, = \rangle \). Countless other choices for atoms include total order atoms \( \langle \mathbb{Q}, \leq \rangle \), or even timed atoms \( \langle \mathbb{Q}, \leq, +1 \rangle \), which make register automata similar (but not identical) to timed ones.

The results shown in the thesis, for the sake of simplicity, are presented within the original regime of equality atoms; however, they can be generalised to other kinds of atoms, the method of this generalisation being described in the relevant chapters.

1.3 Automata with counters (VASS)

*Vector addition systems* \([51]\) (VAS) are a widely accepted model of concurrency equivalent to Petri nets. Another equivalent model, called *vector addition systems with states* (VASS) \([47]\), is an extension of VAS with a finite control graph. It can also be seen as a finite automaton over a unary alphabet with counters, on which the transitions can perform operations of increment or decrement (but no zero tests), with the proviso that counter values are initially set to zero and must remain non-negative along a run. The number of counters \( d \) defines the *dimension* of a VASS, model of dimension \( d \) being denoted as \( \text{VASS}_d \).

One of the main theoretically relevant questions for VASS is the problem of reachability. It can be formulated as follows: given a VASS with two distinguished states \( p, q \), does there exist a run from \( p \) to \( q \), starting and ending with all counters set to zero? The problem’s status has recently been clarified—several decades after the first decidability proof of Mayr \([61]\) was published in 1981. However, the resulting Ackermann-completeness of the problem (dropping to \( \text{PSPACE} \)-completeness for \( \text{VASS}_2 \)) essentially precludes practical applications. For this reason, when working with the difficult to analyse VASS model, one may be tempted to try to seek a simpler description of its properties. One such approach is the question of the regular separability of languages of labelled VASS, which is open already for \( \text{VASS}_2 \).

\(^3\)In fact, even the inclusion problem \( \mathcal{L}(A) \subseteq \mathcal{L}(B) \) with \( A \) an \( \text{nra} \) and \( B \) a DRA is decidable.
Described in the dissertation is a *pumping method* for runs of $\text{VASS}_2$, which we developed with the hope of eventually solving the regular separability problem in the two-dimensional case. Although we did not achieve our ultimate goal, our method—internally making use of a variety of geometric observations—seems to be a step in the right direction.

**Pumping**

Broadly speaking, *pumping* is a class of techniques exploiting repetitions of states in runs. It is a ubiquitous phenomenon which typically provides valuable tools in proving short run properties, characterizing set of runs or showing language inexpressibility results. It seems to be particularly relevant in case of $\text{VASS}$, as even the core of the seminal decision procedure for the reachability problem in $\text{VASS}$ by Mayr and Kosaraju [55, 62] is fundamentally based on pumping. The decision procedure they propose essentially decomposes a $\text{VASS}$ into a finite number of more structured $\text{VASS}$, each having the property that every path can be pumped up so that it induces a run.

Pumping techniques are used even more explicitly when dealing with subclasses of $\text{VASS}$ of bounded dimension. The $\text{PSPACE}$ upper bound for the reachability problem in $\text{VASS}_2$ [9] relies on various un-pumping transformations of an original run, leading to a simple run of at most exponential length, in the form of a short path with adjoined short disjoint cycles. A smart combinatorial manipulation of these simple runs was also used to obtain a stronger upper bound ($\text{NL}$) in case when the transition effects are represented in unary [36]. Un-pumping is also used in [15] to provide a quadratic bound on the length of the shortest run for $\text{VASS}_1$, also known as one counter automata without zero tests, and for unrestricted one counter automata. See also [6, 59] for pumping techniques in one counter automata.

## 2 Contributions grouped by problems

Here, we summarise the main results found in the dissertation. Each of following §§2.1 to 2.4 first sketches the problem and the state of the art on it, and then presents the newly proven theorems.

### 2.1 Synthesis problem and synthesis games

The principal result of the dissertation concerns the previously listed question Q1

**Q1** Given a complex specification, does there exist a simpler device realising it?

(synthesis problem)

where the (complex) specification is specified in terms of $\text{NRA/NTA}$, and the goal is to provide a deterministic device realising it. Before we discuss it, we need to introduce the notions of $\omega$-regular languages and synthesis games.
\section*{ω-regular languages}

For finite automata over alphabet $\Sigma$, there are many ways of adapting them to recognise \textit{infinite words} (from $\Sigma^\omega$) instead of finite ones (from $\Sigma^*$). Some of these approaches only modify semantics, while others introduce subtle alterations of the syntax, too. One may consider \textsc{dra} or \textsc{nra} with updated acceptance conditions, i.a., Büchi, Muller, or parity condition. Most\footnote{All classes but \textsc{dra} equipped with Büchi condition, which can only express a strict subset of \textit{ω-regular} languages.} of these variants have the same set of languages they can express—this way, the class of \textit{ω-regular} languages arises.

\section*{Synthesis games}

One of the areas in which \textit{ω-regular} languages find their use are \textit{synthesis games}. These infinite-duration games are played by two players, Alice and Bob, who both are assigned finite alphabets of actions, $A$ and $B$. An instance of a game is given as its winning set—an \textit{ω-regular} language $W \subseteq (A \times B)^\omega$. Each turn consists of a single action of Alice followed by one of Bob. The completed infinite play is a word $w \in (A \times B)^\omega$, and Alice wins if $w \in W$. A \textit{finite memory strategy} in such a game is a deterministic automaton with additional output\footnote{Other names by which such automaton with output is known include ‘letter-to-letter transducer’, and ‘Mealy machine’.} letter written on every transition. It reads the opponent’s actions one by one and responds to them synchronously, with current player’s actions.

\section*{Synthesis problem}

In their famous result \cite{12}, Büchi and Landweber have shown that one can

\begin{itemize}
  \item decide the winner of such games,
  \item compute (synthesize) a finite-memory strategy for the winner.
\end{itemize}

Intuitively, if a player can win using arbitrary (possibly infinite) memory, they can also win using a finite memory strategy, which can be effectively computed.

We generalise these games to the setting of automata over infinite alphabets, defining \textit{timed synthesis games} and \textit{register synthesis games}. Since the two variants share many similarities, we present them jointly here. The differences are highlighted with square brackets $[X,Y]$, where $X$ and $Y$ concern timed and register variant, respectively.

\section*{Timed and register synthesis games}

There are two players, called Alice and Bob, who take turns in a strictly alternating fashion. At the $i$-th round, Alice selects a letter $a_i$ from a finite alphabet and [a nonnegative timestamp/an atom] $\chi_i$ from $[Q_{\geq 0}/A]$. Bob replies with a letter $b_i$ from a finite alphabet. This process continues infinitely. At the doomsday, the two players will have built an infinite play $\pi = (a_1, b_1, \chi_1)(a_2, b_2, \chi_2) \cdots$, and Alice wins if, and only if, $\pi$ belongs to her winning set $W$, which is specified as the language of a \textit{[NTA/NRA]} (possibly with $\varepsilon$-transitions).

Analogously to original synthesis games, finite-memory strategies in above game correspond to \textit{[timed/register]} automata with output.
TIMED AND REGISTER SYNTHESIS PROBLEMS

The \textit{[timed/register] synthesis problem} is to compute, if it exists, a winning controller for Bob —\([\text{DTA/TRA}]\) with output—which ensures that every play \(\pi\) conformant to that controller is winning for Bob. We also consider two more restricted variants of it. For a fixed number of \([\text{clocks/registers}]\) \(k \in \mathbb{N}\), the \textit{[k-clock timed/k-register] synthesis problem} asks—as before—about a controller for Bob, but this time using at most \(k\) \([\text{clocks/registers}]\). Finally, exclusively for the timed variant, we define a \textit{k-clock m-constrained timed synthesis problem}, which is analogous to \(k\)-clock timed synthesis, but also requires the controller’s transitions only to feature constraints with constant bounded by \(m\) in absolute value. With all problems, we naturally associate decision problems (‘does a winning controller exist?’).

We show the computability of the restricted variants of synthesis problem, which are analogues to the Büchi-Landweber result.

\begin{shaded}
\begin{theorem}
For every fixed \(k \in \mathbb{N}\), the \(k\)-register synthesis problem is computable.
\end{theorem}
\end{shaded}

\begin{shaded}
\begin{theorem}
For every fixed \(k, m \in \mathbb{N}\), the \(k\)-clock \(m\)-constrained timed synthesis problem is computable.
\end{theorem}
\end{shaded}

\begin{shaded}
\begin{theorem}
For every fixed \(k \in \mathbb{N}\), the \(k\)-clock timed synthesis problem is computable.
\end{theorem}
\end{shaded}

The main part of our proof is the reduction of these games to finite-state games with an \(\omega\)-regular winning conditions. This is done via introduction of a special protocol using orbits/regions. Then, they can be solved using the seminal result of Büchi and Landweber [12].

It is important to note that the \(k\)-clock timed synthesis problem requires the synthesis of the maximal constant \(m\), which is a technical novelty not shared with the current literature on timed games. This result is obtained, intuitively, through a design of protocol in which Bob demands from Alice to be informed when clocks elapse one time unit. We require the number of such consecutive requests to be finite, yielding a bound on \(m\) (when such a value exists).

Additionally, our construction of synthesis games in itself is noteworthy, as we thoughtfully designed them to provide high expressiveness, while avoiding unnecessary complication:

- It was our informed choice to only allow Alice to use an infinite alphabet of actions. This way, finite-memory strategies of Bob in our games are equivalent to timed/register automata with output symbols (a.k.a. letter-to-letter transducers).

- Additionally, it is important that we specify the winning condition for Alice. In fact, the same problem—with the set of winning plays for Bob specified by a nondeterministic timed language—becomes uncomputable (cf. [35] for a similar observation).
When compared with our setup, variant of register synthesis problem considered by Khalimov et al. [52] (settled independently) may ostensibly seem more expressive. In their framework—not specified in terms of game—the transducer (automaton with output) can also output atoms. In our terminology, this situation corresponds to a game, in which both Alice and Bob can play atoms, but Bob is limited only to play these which Alice has used so far. However, already in our original, simpler setting, one can obtain undecidability of register synthesis problem.

**Theorem 2.4.**  

The register synthesis decision problem is undecidable, and this holds already when Alice’s winning condition is an \( \text{NRA}_2 \) language.

and of timed synthesis, too:

**Theorem 2.5.**  

The timed synthesis decision problem is undecidable, and this holds already when Alice’s winning condition is an \( \text{NTA}_2 \) language.

Moreover, as in [52] the transducer is limited to use only atoms stored in registers, in context of \([k\text{-clock/}k\text{-register}]\) synthesis problem, his set of choices of symbols is in fact always bounded by \( k \). This makes it amenable to simulation with a finite alphabet. It follows that our \( k \)-register synthesis setup is equally expressive as one of [52], while additionally allowing other atom domains.

### 2.2 Deterministic separability problem

Separability is a classical problem in theoretical computer science and mathematics. For sets \( S, L, M \), we say that \( S \) **separates** \( L \) from \( M \) if \( L \subseteq S \) and \( S \cap M = \emptyset \).

Intuitively, a separator \( S \) provides a certificate of disjointness, yielding information on the structure of \( L, M \) up to some level of granularity.

There are many elegant results in computer science and mathematics showing that separators with certain properties always exist, such as Lusin’s separation theorem in topology (two disjoint analytic sets are separable by a Borel set), Craig’s interpolation theorem in logic (two contradictory first-order formulas can be separated by one containing only symbols in the shared vocabulary), in model theory (two disjoint projective classes of models are separable by an elementary class), in formal languages (two disjoint Büchi languages of infinite trees are separable by a weak language, generalising Rabin’s theorem [72]), in computability (two disjoint co-recursively enumerable sets are separable by a recursive set), in the analysis of infinite-state systems (two disjoint languages recognisable by well-structured transition systems are regular separable [27]), etc.

When separability is not trivial, one may ask whether existence of separator is decidable. Let \( \mathcal{C} \) and \( \mathcal{S} \) be two classes of sets. The **\( \mathcal{S} \)-separability** problem for \( \mathcal{C} \) amounts to decide whether, for every input sets \( L, M \in \mathcal{C} \) there is a set \( S \in \mathcal{S} \) separating \( L, M \). Many results of this kind exist when \( \mathcal{C} \) is the class of regular languages of finite words over finite alphabets, and \( \mathcal{S} \) ranges over piecewise-testable languages [29,68] (later generalised...
2. Contributions grouped by problems

to context-free languages [30] and finite trees [43]), locally and locally threshold testable languages [69], first-order logic definable languages [71] (generalised to some fixed levels of the first-order hierarchy [70]). For classes of languages $\mathcal{C}$ beyond the regular ones, decidability results are more rare. For example, regular separability of context-free languages is undecidable [48, 54, 75]. Nonetheless, there are positive decidability results for separability problems on several infinite-state models, such as Petri nets [19], Parikh automata [17], one-counter automata [25], higher-order and collapsible pushdown automata [23, 45], and others.

In this dissertation, we study the separability problem for languages of timed and register automata. In particular, we are interested in the deterministic separability problem, in which we seek a simple—deterministic—separator for languages of more complex—nondeterministic—devices. This matches the previously stated question Q2:

**Q2** Can disjoint languages of complex models be distinguished (separated) in a simpler way? (separability problem)

More precisely, the problem asks, given two nondeterministic automata $A$ and $B$ with disjoint languages, be it timed or register ones, whether there exists a deterministic automaton $S$ such that $L(S)$ separates $L(A)$ from $L(B)$. Various ways of quantifying the models’ parameters, like the desired number of clocks/registers of the separator, give rise to the following variants of the separability question:

- The $k$-clock $m$-constrained deterministic separability problem for timed automata asks whether $A, B \in \text{nta}$ can be separated by a deterministic automaton $S \in \text{dta}$ with $k$ clocks measuring up to $m$ units of time.

- The $k$-clock deterministic separability problem for timed automata is defined similarly, but with the restraint on $m$ is lifted. This way the resulting separator $S$ can measure the time up to arbitrary fixed number of time units.

- Finally, the $k$-register deterministic separability for register automata asks whether $A, B \in \text{nra}$ can be separated by a deterministic automaton $S \in \text{dra}$ with $k$ registers.

Intuitively, deterministic separability problem strengthens disjointness by additionally requiring a simple (deterministic) reason of disjointness of more complex (nondeterministic) models.

We can also see $A$ as recognising a set of good behaviours which we want to preserve and $B$ recognising a set of bad behaviours which we want to exclude. From this point of view, a deterministic separator, when it exists, provides a kind of compromise between these two conflicting requirements.

Concerning separability, our main contribution is decidability of $k$-clock $m$-constrained and $k$-clock deterministic separability, obtained by means of synthesis games discussed in §2.1. We provide analogous results for both timed and register automata:

**Theorem 2.6.** \(\text{separability by dra}_k\)

For $k \in \mathbb{N}$, the $k$-register deterministic separability problem for $\text{nra}$ is decidable.
To the best of our knowledge, separability problems for timed and register automata have not been investigated before.

Decidability of timed deterministic separability should be contrasted with undecidability of the corresponding deterministic membership problem—the question whether a nondeterministic timed language is deterministic is undecidable [38, 78] (also cf. §2.3). This is a rare circumstance, which is shared with languages recognised by one-counter nets [25], and conjectured to be the case for the full class of Petri net languages.

Similar situation holds also for register automata, as we outline in the next section §2.3.

### 2.3 Deterministic membership problem

Here, we consider an instance of the question Q3:

**Q3** Can a complex device be equivalently described in a simpler way?

\[\text{(membership problem)}\]

where ‘complex’ and ‘simpler’ mean ‘nondeterministic’ and ‘deterministic’, respectively. We provide analogous results for both timed and register automata.

**Timed automata**

The \(\text{DTA} \) membership problem asks, given an \(\text{NTA} \), whether there exists a \(\text{DTA} \) recognising the same language. There are two natural variants of this problem, which are obtained by restricting the resources available to the sought \(\text{DTA} \). Let \(k \in \mathbb{N}\) be a bound on the number of clocks, and let \(m \in \mathbb{N}\) be a bound on the maximal absolute value of numerical constants.

- The \(\text{DTA}_k\) is a variant of the problem above where the \(\text{DTA} \) is required to have at most \(k \) clocks.
- The \(\text{DTA}_{k,m}\) membership problem asks for the automaton with at most \(k \) clocks and constants bounded by \(m \) in absolute value.

\[\text{6}\] All these classes of languages have a decidable disjointness problem, however regular separability is not always decidable in this case [77].
Notice that we do not bound the number of control locations of the $\text{DTA}$, which makes the problem non-trivial. (Indeed, there are finitely many $\text{DTA}$ with a bounded number of clocks, control locations, and maximal constant.)

Since untimed regular languages are deterministic, the $\text{DTA}_k$ membership problem can be seen as a quantitative generalisation of the regularity problem. For instance, the $\text{DTA}_0$ membership problem is precisely the regularity problem since a timed automaton with no clocks is essentially the same as a finite automaton. We remark that the regularity problem is usually undecidable for nondeterministic models of computation generalising finite automata, e.g., context-free grammars/pushdown automata [73, Theorem 6.6.6], labelled Petri nets under reachability semantics [80], Parikh automata [13], etc. One way to obtain decidability is to either restrict the input model to be deterministic (e.g., [7, 79, 80]), or to consider more refined notions of equivalence, such as bisimulation (e.g., [42]).

This negative situation is generally confirmed for timed automata. For every number of clocks $k \in \mathbb{N}$ and maximal constant $m$, the $\text{DTA}$, $\text{DTA}_k$, and $\text{DTA}_{k,m}$ membership problems are known to be undecidable when the input $\text{NTA}$ has $\geq 2$ clocks, and for 1-clock $\text{NTA}$ with epsilon transitions [38, 78]. To the best of our knowledge, the deterministic membership problem was not studied before when the input automaton is $\text{NTA}_1$ without epsilon transitions.

**REGISTER AUTOMATA**

The situation with register automata is similar to that of timed automata, only simpler.

- The $\text{DRA}_k$ membership problem asks, given an $\text{NRA}$, whether there exists a $\text{DRA}$ with $k$ registers recognising the same language.

- The $\text{DRA}$ membership problem is the same problem, but with no apriori bound on the number of registers of the deterministic acceptor.

Deterministic membership problems for register automata do not seem to have been considered before in the literature.

**CONTRIBUTIONS**

We complete the study of the decidability border for the deterministic membership problem initiated for timed automata in [38, 78], and we extend these results to register automata.

**UPPER BOUNDS**

Our main results on the deterministic membership problem for timed automata are

**Theorem 2.9.** $\text{DTA}_k$ membership

For every fixed $k \in \mathbb{N}$, the $\text{DTA}_k$ membership problem is decidable for $\text{NTA}_1$ languages.

**Theorem 2.10.** $\text{DTA}_{k,m}$ membership

For every fixed $k, m \in \mathbb{N}$, the $\text{DTA}_{k,m}$ membership problem is decidable for $\text{NTA}_1$ languages.
Our decidability result contrasts starkly with the abundance of undecidability results for the regularity problem. We establish decidability by showing that if an $\mathsf{NTA}_{1,m}$ recognises a $\mathsf{DTA}_k$ language, then, in fact, it recognises a $\mathsf{DTA}_{k,m}$ language and, moreover, there is a computable bound on the number of control locations of the deterministic acceptor. This provides a decision procedure since there are finitely many different $\mathsf{DTA}$ once the number of clocks, the maximal constant, and the number of control locations are fixed.

In our technical analysis, we find it convenient to introduce the so-called *always resetting* subclass of $\mathsf{NTA}_k$. These automata are required to reset at least one clock at every transition and are thus of expressive power intermediate between $\mathsf{NTA}_{k-1}$ and $\mathsf{NTA}_k$. Always resetting $\mathsf{NTA}_2$ are strictly more expressive than $\mathsf{NTA}_1$: For instance, the language of timed words of the form $(a, \tau_0)(a, \tau_1)(a, \tau_2)$ such that $\tau_2 - \tau_0 > 2$ and $\tau_2 - \tau_1 < 1$ can be recognised by an always resetting $\mathsf{NTA}_2$ but by no $\mathsf{NTA}_1$. Despite their increased expressive power, always resetting $\mathsf{NTA}_2$ still have a decidable universality problem (the well-quasi order approach of [66] goes through), which is not the case for $\mathsf{NTA}_2$. Thanks to this restricted form, we are able to prove a lemma providing an elegant characterisation of those $\mathsf{NTA}_1$ languages which are recognised by an always resetting $\mathsf{DTA}_k$.

We prove a result analogous to theorem 2.9 in the setting of register automata.

### Theorem 2.11

For every fixed $k \in \mathbb{N}$, the $\mathsf{DRA}_k$ membership problem is decidable for $\mathsf{NRA}_1$ languages.

Thanks to the effective elimination of $\varepsilon$-transition rules from $\mathsf{NRA}_1$, the decidability result above also holds for data languages presented as $\mathsf{NRA}_1$ with $\varepsilon$-transition rules.

### LOWER BOUNDS

We complement the decidability results above by showing that the deterministic membership problem becomes undecidable if we do not restrict the number of clocks/registers of the deterministic acceptor.

### Theorem 2.12

The $\mathsf{DTA}$ and $\mathsf{DTA}_{*,m}$ membership problems are undecidable for $\mathsf{NTA}_1$ without epsilon transitions.

Theorem 2.12 is shown by improving on an analogous result from [38, Theorem 1] for $\mathsf{NTA}_2$. We obtain a similar undecidability result in the setting of register automata:

### Theorem 2.13

The $\mathsf{DRA}$ membership problem is undecidable for $\mathsf{NRA}_1$.

The following lower bounds further refine the analysis from [38] in the case of a fixed number of clocks of a deterministic acceptor.
2. Contributions grouped by problems

Theorem 2.14. **Undecidability and hardness for $\text{DTA}_k$ membership**

For every fixed $k, m \in \mathbb{N}$, the $\text{DTA}_k$ and $\text{DTA}_{k,m}$ membership problems are:

1. undecidable for $\text{NTA}_2$,
2. undecidable for $\text{NTA}_1^\varepsilon$ (NTA with epsilon transitions),
3. HyperAckermann-hard for $\text{NTA}_1$.

A similar landscape holds for register automata, where the deterministic membership problem for a fixed number of registers of the deterministic acceptor remains undecidable when given in input either an $\text{NRA}_2$ or an $\text{NRA}_1$ with guessing.\(^7\). In the decidable case of an $\text{NRA}_1$ input, the problem is nonetheless not primitive recursive.

Theorem 2.15. **Undecidability and hardness for $\text{DRA}_k$ membership**

Fix a $k \geq 0$. The $\text{DRA}_k$ membership problem is:

1. undecidable for $\text{NRA}_2$,
2. undecidable for $\text{NRA}_1^\varnothing$ (NRA with guessing), and
3. not primitive recursive (Ackermann-hard) for $\text{NRA}_1$.

2.4 Pumping technique for VASS of dimension two

The pumping techniques mentioned in §1.3 are mostly oriented towards reachable sets, and henceforth may ignore certain runs as long as the reachable set is preserved. As consequence, they are not very helpful in solving decision problems formulated in terms of the language accepted by a VASS, like the regular separability problem (cf. [18, 28, 32]).

In the dissertation, we restrict ourselves exclusively to the VASS of dimension two, our primary objective being to design means of pumping applicable to every run of a VASS\(_2\). Therefore, as our main technical contribution related to VASS\(_2\), we perform a meticulous classification of runs, in form of a dichotomy:

For every run $\pi$ of a VASS\(_2\) starting and ending with both counters set to zero,

- either $\pi$ is *thin*, by which we mean that the counter values along the run stay within belts, whose direction and width are all bounded polynomially in the number $n$ of states and the largest absolute value $M$ in vectors of the VASS\(_2\);
- or $\pi$ is *thick*, by which we mean that a number of cycles is enabled along the run, and the effect vectors of these cycles, roughly speaking, span the whole plane. Additionally, the lengths of cycles and the initial and final factors of $\pi$ are all bounded polynomially in $M$ and exponentially in $n$.

\(^7\)Register automata with guessing are a more expressive family of automata where a register can be updated with a data value not necessarily coming from the input word, i.e., it can be *guessed*. 

14
The precise formulation of the dichotomy, omitting the definitions of thin and thick runs, is as follows. It speaks of \((0,0)\)-runs, i.e., runs starting and ending with both counters set to 0.

**Theorem 2.16.** thin/thick dichotomy

There is a polynomial \(p\) such that every \((0,0)\)-run in a \(\text{VASS}_2\) \(V\) is either \(p(nM)^n\)-thin or \(p(nM)^n\)-thick.

The dichotomy immediately entails a pumping theorem for \(\text{VASS}_2\).

**Theorem 2.17.** pumping runs of \(\text{VASS}_2\)

There is a polynomial \(p\) such that every \((0,0)\)-run \(\tau\) in a \(\text{VASS}_2\) of length greater that \(p(nM)^n\) factors into \(\tau = \tau_0 \tau_1 \ldots \tau_k\) \((k \geq 1)\), so that for some non-empty cycles \(\alpha_1, \ldots, \alpha_k\) of length at most \(p(nM)^n\), the path \(\tau_0 \alpha_1^i \tau_1 \alpha_2^i \ldots \alpha_k^i \tau_k\) is a \((0,0)\)-run for every \(i \in \mathbb{N}\). Furthermore, the lengths of \(\tau_0\) and \(\tau_k\) are also bounded by \(p(nM)^n\).

Intuitively, adapted pumping scheme of \(\text{VASS}_1\) is used in case of thin runs, whereas for thick runs we utilize the collected cycles to construct correct ‘inflated’ runs.

As a more subtle application of the dichotomy, we derive an alternative proof of the exponential run property (shown originally in [9]), which immediately implies \(\text{PSPACE}\)-membership of the reachability problem.

**Theorem 2.18.** exponential run property

There is a polynomial \(p\) such that for every \((0,0)\)-run \(\tau\) in a \(\text{VASS}_2\), there is a \((0,0)\)-run of length bounded by \(p(nM)^n\) with the same source and target as \(\tau\).

### 3 Source materials

Many of the results presented in the thesis stem from the author’s close collaboration with several people. This has borne fruit in scientific articles, already published or awaiting publication. Other results of this collaboration, previously unpublished, are presented for the first time. The list below summarises the relationship between the source materials and theorems in this thesis.

**Sources**

1. *Timed Games and Deterministic Separability* [22] (ICALP 2020)
   Authors: Lorenzo Clemente, Sławomir Lasota and P.
   - Theorems 2.2, 2.3 and 2.5 (timed synthesis)
   - Theorems 2.7 and 2.8 (DTA separability)

2. *Determinisability of One-Clock Timed Automata* [21] (CONCUR 2020)
   Authors: Lorenzo Clemente, Sławomir Lasota and P.
   - Theorems 2.9, 2.10 and 2.14 (DTA membership)
4. Conclusion

In this dissertation, we present an extensive set of results that fall within the general quest for simplifying more complex systems. We propose a relatively simple framework of register and timed synthesis games—a generalisation of the Church’s synthesis problem to infinite-state systems. The computability results we obtain for controllers with a limited number of registers or clocks allow us to easily settle the corresponding variants of the deterministic separability problem. Unusually, this solution is common to both models. Unusually—as, often, in spite of their far-reaching similarities, timed and register automata generate inconsistencies which make it impossible to apply uniform proof techniques. Our set of results is completed by the definite solution to the deterministic membership problem (resolving one-register/clock corner cases), and by the introduction of a run pumping technique based on geometric observations, with potential application in proving further results for $\text{VASS}$.

Open Problems

While working on the results of this dissertation, we encountered several problems which have not yet been resolved. These include:

- decidability status of $\text{DRA}$- and $\text{DTA}$-separability problem
  Unlike the corresponding synthesis result, separability resists an undecidability proof because of its simpler, more constraining structure.

- decidability status of regular separability for $\text{VASS}_2$
  So far, the only positive results obtained in this area assume one of the separated languages belongs to $\text{VASS}_1$ (cf. [31]). This is the simplest as yet unsolved case of the general regular separability problem of languages of $\text{VASS}_k$.

We leave these questions open as a set-up for further research work.

3. Determinisability of register and timed automata (submitted to LMCS)
Authors: Lorenzo Clemente, Sławomir Lasota and P.
- theorems 2.9 to 2.11, 2.14 and 2.15 ($\text{DTA}$ and $\text{DRA}$ membership)

4. New Pumping Technique for 2-Dimensional $\text{VASS}$ [26] (MFCS 2019)
Authors: Wojciech Czerwinski, Sławomir Lasota, Christof Löding and P.
- theorems 2.17 and 2.18 (pumping technique for $\text{VASS}_2$)

5. Previously unpublished results
Authors: Lorenzo Clemente, Sławomir Lasota and P.
- theorems 2.1 and 2.4 (register synthesis)
- theorem 2.6 ($\text{DRA}$ separability)
BIBLIOGRAPHY


