University of Warsaw
Faculty of Mathematics, Informatics and Mechanics

Radosław Piórkowski

Tame the infinite — Simplification problems for infinite-state systems

PhD dissertation
in COMPUTER SCIENCE

Supervisor:
prof. dr hab. Sławomir Lasota
Institute of Informatics
Auxiliary supervisor:
dr hab. Wojciech Czerwiński
Institute of Informatics

Warsaw, April 2022

Author's declaration
This dissertation is ready to be reviewed.

20 April 2022

Radosław Piórkowski

Supervisor's declaration
This dissertation is ready to be reviewed.

20 April 2022

prof. dr hab. Sławomir Lasota

Auxiliary supervisor's declaration
This dissertation is ready to be reviewed.

20 April 2022

dr hab. Wojciech Czerwiński

Abstract

Centred on models with an infinite state space, the thesis explores computational problems that fall within the general notion of 'simplification'. This includes examining—with a particular use case in mind—under what conditions a simple model can suffice to replace a complex one. We obtain results for register (RA) and timed automata (TA), as well as for vector addition systems (VASS).

Synthesis and games The first of the problems analysed is Church's synthesis in game-theoretic terms, which we generalise to cover register and timed setting. Our game models interactions between a deterministic controller (Bob) and its operating environment (Alice), which includes atoms or timestamps; only Alice's set of actions is infinite. A winning condition—given by a nondeterministic RA with guessing or a nondeterministic TA —specifies when Alice wins. In both settings, we show computability of Bob's winning resource-limited controller, if it exists. More precisely, the problem is solved when the input provides a limit on ▶ the number of registers, ▶ both the number of clocks and the maximal numerical constant available to Bob, ▶ just the number of clocks. Interestingly, the last variant does not require the constant to be given in advance, which is an important technical novelty not present in previous literature on timed games. We complement these results by showing undecidability when the resources of Bob are not fixed.

Deterministic separability A set separates two others when it contains one of them and is disjoint from the other. Deterministic separability is a novel decision problem about register and timed automata which has not been studied before. It asks whether two languages of nondeterministic models are separated by a language of a deterministic one. We reduce separability to resource-bounded synthesis by defining a game that characterises separability. By means of it, the synthesis results translate into solutions to the separability problem, the type of winning condition coinciding with the class of languages to be separated, and the separator being subject to the same resource constraints as the controller. Whether separability is decidable without limiting the number of separator registers/clocks remains an interesting open problem.

Deterministic membership problems The deterministic membership problem for TA asks whether, for a given nondeterministic TA, there exists a deterministic one having the same language. An analogous problem is also stated for register automata. We draw the complete decidability/complexity landscape for both variants of the problem. For register automata, we prove that the deterministic membership problem is decidable when the input automaton is a nondeterministic one-register automaton without guessing and the number of registers of the output deterministic register automaton is fixed. This is optimal, as we show that omitting any of the three assumptions about the output or input automaton results in undecidability. In the timed setting, the situation is similar. We show that the problem is decidable when the input automaton is a one-clock nondeterministic timed automaton without epsilon transitions and the number of clocks of the output deterministic timed automaton is fixed. Again, we prove this to be optimal.

Utility results for VASS A vector addition system is similar to a finite automaton equipped with counters, which are not zero-testable. We propose a new pumping technique for two-dimensional vector addition systems with states (2-VASS) building on natural geometric properties of runs. We illustrate its applicability by reproving an exponential bound on the length of the shortest accepting run, and by proving a new pumping lemma for languages of 2-VASS. The technique is expected to be useful for settling questions concerning languages of 2-VASS, e.g., for establishing decidability status of the regular separability problem.

Keywords

theory of computation, infinite-state models of computation, timed automata, register automata, vector addition systems, decidability, Church synthesis, separability, determinisability, pumping

Thesis domain (Socrates-Erasmus subject area codes)

11.3 Computer Science

Subject classification

Theory of computation Automata over infinite objects
Theory of computation Quantitative automata
Theory of computation Timed and hybrid models

Tytuł pracy w języku polskim

Okiełznać nieskończone—problemy upraszczania dla systemów nieskończenie stanowych

Contents

List of theorems . . . 7
Acknowledgements . . . 11
Tale . . . 13

1. Overview . . . 17
1.1. Models of computation . . . 18
1.1.1. Automata with clocks (NTA/DTA) . . . 19
1.1.2. Automata with registers (NRA/DRA) . . . 20
1.1.3. Automata with counters (VASS) . . . 21
1.2. Contributions grouped by problems . . . 22
1.2.1. Synthesis problem and synthesis games . . . 22
1.2.2. Deterministic separability problem . . . 25
1.2.3. Deterministic membership problem . . . 27
1.2.4. Pumping technique for VASS_2 . . . 30
1.3. Source materials . . . 31

2. Preliminaries . . . 33
2.1. Models of computation with infinite space of configurations . . . 33
2.2. Labelled transition systems (LTS) . . . 34
2.3. Sets with atoms . . . 40
2.3.1. Atoms . . . 40
2.3.2. Atom automorphisms . . . 42
2.3.3. Invariance . . . 43
2.4. Register automata (RA) . . . 44
2.4.1. Register constraints . . . 45
2.4.2. Nondeterministic register automata with guessing ($\mathsf{NRA}^{\mathsf{g}}$) . . . 46
2.4.3. Nondeterministic register automata without guessing (NRA) . . . 48
2.4.4. Deterministic register automata (DRA) . . . 49
2.4.5. One-register automata . . . 50
2.4.6. Invariance of register automata . . . 50
2.4.7. Varieties of register automata . . . 51
2.5. Timed automata (TA) . . . 52
2.5.1. Timed words and languages . . . 53
2.5.2. Clock constraints and regions . . . 53
2.5.3. Nondeterministic timed automata (NTA) . . . 54
2.5.4. Deterministic timed automata (DTA) . . . 56
2.6. Vector addition systems with states (VASS) . . . 57
2.7. Computational problems . . . 59

2.7.1. Synthesis problem and synthesis game 59
2.7.2. Deterministic separability problems 61
2.7.3. Deterministic membership problems 62

3. Generalised synthesis problems and games 65
3.1. Generalised deterministic LTS synthesis game 65
3.2. Register synthesis game 67
3.3. Timed synthesis game 68

4. Synthesis game for solving deterministic separability 71
4.1. Motivating examples 71
4.2. Reduction to generalised synthesis problem 72

5. Solving k-register synthesis problem 75
5.1. Atom-blind synthesis problem 76
5.2. Eliminating winning controller's registers 76
5.3. Eliminating atoms 79

6. Solving k-clock timed synthesis problems 81
6.1. Simplifying assumptions 82
6.1.1. Zero-starting winning conditions 82
6.1.2. Strictly monotonic winning conditions 83
6.2. Elliminating winning controller's clocks 84
6.2.1. Case of bounded constants in clock constraints 84
6.2.2. Case of unbounded constants in clock constraints 90
6.3. Elliminating time 92

7. Solving deterministic membership problems 95
7.1. Decidability of DRA_1 membership for NRA_1 95
7.1.1. Proof of lemma 7.2 96
7.1.2. Other atoms 102
7.2. Lower bounds for NRA 102
7.2.1. Invariance of timed automata 104
7.3. Decidability of DTA_1 membership for NTA_1 108
7.3.1. Proof of lemma 7.30 110
7.4. Lower bounds for NTA 116

8. Undecidability in case of unbounded number of registers/clocks . . 119
8.1. Lossy counter machines 120
8.2. Representing LCM runs as words 120
8.3. Representing LCM runs as timed words 122
8.4. Undecidability of DTA membership 124
8.5. Undecidability of timed synthesis 129
8.5.1. Definition of $\mathsf{G}^{\mathbb{T}}_{\mathcal{M}}$ 130
8.5.2. Intuitition 130
8.5.3. Proofs 132
8.6. Representing LCM runs as data words 134
8.7. Undecidability of DRA membership 138

8.8. Undecidability of register synthesis 142
8.8.1. Local errors and local correctness 143
8.8.2. Definition of $\mathsf{G}^{\mathbb{A}}_{\mathcal{M}}$ 143
8.8.3. Proofs 144
8.8.4. Eliminating guessing 146

9. Utility results for VASS_2 147
9.1. Supporting definition – sequential cones 147
9.2. Thin/thick dichotomy of runs 150
9.2.1. Proof of theorem 9.7 152
9.2.2. Proof of lemma 9.8 155
9.3. Dichotomy at work – pumping lemma & short run property 159
9.3.1. Proof thin 160
9.3.2. Proof thick 163

10. Closing remarks 167

Bibliography 169

List of theorems

Synthesis games

Positive results

Theorem 6.1 synthesis of $\mathsf{DTA}_{k,m}$

For every fixed $k, m \in \mathbb{N}$, the k-clock m-constrained timed synthesis problem is computable.

Theorem 6.2 synthesis of DTA_k

For every fixed $k \in \mathbb{N}$, the k-clock timed synthesis problem is computable.

Theorem 5.1 synthesis of DRA_k

For every fixed $k \in \mathbb{N}$, the k-register synthesis problem is computable.

Undecidability results

Theorem 8.2 synthesis of DTA und.

The timed synthesis decision problem is undecidable, and this holds already when Alice's winning condition is an NTA_2 language.

Theorem 8.4 synthesis of DRA und.

The register synthesis decision problem is undecidable, and this holds already when Alice's winning condition is an NRA_2 language.

Deterministic separability

Theorem 4.2 separability by DTA_k

For $k \in \mathbb{N}$, the k-clock deterministic separability problem for NTA are decidable.

Theorem 4.3 separability by $\mathsf{DTA}_{k,m}$

For $k, m \in \mathbb{N}$, the k-clock m-constrained separability problem for NTA is decidable.

Theorem 4.1 separability by DRA_k

For $k \in \mathbb{N}$, the k-register deterministic separability problem for NRA is decidable.

Deterministic membership

Positive results

Theorem 7.28 DTA_k membership

For every fixed $k \in \mathbb{N}$, the DTA_k membership problem is decidable for NTA_1 languages.

Theorem 7.29 $\mathsf{DTA}_{k,m}$ membership

For every fixed $k, m \in \mathbb{N}$, the $\mathsf{DTA}_{k,m}$ membership problem is decidable for NTA_1 languages.

Theorem 7.1 DRA_k membership

For every fixed $k \in \mathbb{N}$, the DRA_k membership problem is decidable for NRA_1 languages.

Undecidability and hardness

Theorem 8.3 DRA membership und.

The DRA membership problem is undecidable for NRA_1.

Theorem 8.1 DTA and $\mathsf{DTA}_{\bullet,m}$ membership und.

The DTA and $\mathsf{DTA}_{\bullet,m}$ $(m > 0)$ membership problems are undecidable for NTA_1 without epsilon transitions.

Theorem 7.17 undecidability and hardness for DRA_k membership

Fix a $k \geq 0$. The DRA_k membership problem is:

1. undecidable for NRA_2,
2. undecidable for NRA_1^g (NRA_1 with guessing), and
3. not primitive recursive (Ackermann-hard) for NRA_1.

Theorem 7.48 undecidability and hardness for DTA_k membership

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

1. undecidable for NTA_2,

2. undecidable for $\mathsf{NTA}_1^{\varepsilon}$ (NTA with epsilon transitions),
3. HyperAckermann-hard for NTA_1.

Pumping technique for VASS_2

Theorem 9.7 — thin/thick dichotomy

There is a polynomial p such that every $(0,0)$-run in a VASS_2 V is either $p(nM)^n$-thin or $p(nM)^n$-thick.

Theorem 9.21 — pumping runs of VASS_2

There is a polynomial p such that every $(0,0)$-run τ in a 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 τ_0 and τ_k are also bounded by $p(nM)^n$.

Theorem 9.22 — exponential run property

There is a polynomial p such that for every $(0,0)$-run τ in a VASS_2, there is a $(0,0)$-run of length bounded by $p(nM)^n$ with the same source and target as τ.

Acknowledgements

This dissertation symbolically completes a certain period of the author's life. It was a time rich in unforeseen events ranging in scale from personal to global—at times unfavourable and disheartening, at times uplifting and inspiring. As trivial as it may sound, it was an excellent opportunity for growth, both academic and personal—an important step for me towards becoming an independent scientist, but not only that. I would venture to say that it has also made me a better person—more aware of my own goals, aspirations and motivations in life, but also of my own limitations. How true this complacent claim is, only the future will tell.

I am fortunate to have been able to spend my time as a doctoral student surrounded by the most wonderful people one can imagine, welcoming and understanding, on whom I could rely. It is hard for me to believe my luck: it is generally hard to find kind and friendly people, while I have been surrounded by them all around. This short note cannot fit all my thanks, but I am grateful to everyone for the kindness shown to me. At this point, however, I would like to thank in particular a few people who have supported me most during my studies and later, during the writing of the dissertation.

I am beyond grateful to my PhD advisor, prof. Sławomir Lasota, for the countless hours of discussions we shared both in his faculty room and online, for his willingness to share the vastness of his knowledge, for gently but thoughtfully guiding my development, and finally for the fact that I could always count on his support.

Furthermore, I would like to thank the people with whom I have had the opportunity to collaborate in solving various research problems, which has often resulted in publications. In particular, I thank to dr. hab. Wojciech Czerwiński (who became also my auxiliary supervisor), and to dr. hab. Lorenzo Clemente. Working with you has been both enjoyable and inspiring, I have learned a lot from you.

Additionally, I would also like to thank dr. Piotr Hofman, for the kindness and support shown to me at the beginning of my doctoral studies, which made it incredibly easy for me to enter the scientific world, despite the personal problems I suffered at the time.

I would like to thank those of my friends who have kept my spirits up when I lacked confidence.

Lastly, I would like to thank my parents, with whose support I survived two disasters—pandemics, and writing the doctoral thesis.

Tale

Once upon a time there was[1] a magical realm called the land of computation. Living there was Bob—a humble and kind adventurer with a big heart. A free spirit, going from town to town, he took a liking in solving computational problems, and never walked past those in need.

One day, he was traversing through a small village, when an upsetting sight caught his attention. In the middle of a horrible mess stood a distraught-looking old woman. Surrounded with remains of what must have been a universal Turing machine a few minutes ago, she wrung her hands in a gesture of helplessness. The tape got all twisted and torn; transition rules lying in disarray, some having crushed in pieces; what is worse, the brass head has also broken in two.

“I am doomed” said the lady. “I was just returning from my appointment at tape cleaner’s when, I know not how, the machine slipped out of my hand, and shattered on the ground. I cannot afford another one, while the nearest repair workshop is in the city beyond seven mountains and seven rivers.”

“Let me see, maybe there is something I can do?” Bob offered.

After all, he had quite a bit of experience with various models of computation. Young man collected the scattered transition rules into a set; replacing missing or broken ones with copies carved out of wood. Then, he cut off the ragged and ripped initial fragment of the tape. Fortunately, the roll with the infinite suffix has survived: he only had to wipe it clean from dust and mud. Finally, he took the two brass pieces and looked at them with concern. Sadly, the head was broken beyond repair. Bob sighed quietly, then reached deep into the inside pocket of his coat. He took out a bundle which held his last spare head. A moment later, he handed the assembled machine back to the woman. She looked astonished, touched and immensely relieved.

“Thank you, good man!” said she. “You saved me. Had it not been for your help, I would’ve been crushed by the pile-up of computational problems. I have nothing repay you, but, as a token of my gratitude, please accept this little amethyst. It is not an ordinary stone. When my mother passed it onto me, she revealed that it can help its owner in moments of trial. I was not able to work out its secret, but I believe you will find a way to use it in your time of need.”

The adventurer thanked the woman cordially, though deep down he was doubtful if the purple gemstone he was given had anything more to it than its plain appearance would suggest. Then, they talked for a bit over a hot tea, but shortly after Bob was on his way again.

* * *

[1]It did not really happen.

Not many weeks later, Bob saw from afar something resembling a large white mountain. As he came closer, he saw a town at the foot of what turned out to be a gigantic glossy white-tinted crystal. Since the sun was already setting, he decided to arrange an overnight stay at a one of the town's inns. Having paid the innkeeper, Bob inquired him about the unusual landmark.

"It has been here for as long as I remember" replied the man. "The legend has it that the mountain holds a castle of a princess named Alice. They say that one day, a malicious sorcerer had sneakily planted an innocent-looking gift in the castle: a two-counter machine. When the unsuspecting princess activated it, an overpowering undecidability spell escaped, locking the entire castle in a crystal of never ending computation. Since then, many have tried to save her, whether for love or for the supposed riches of her kingdom, but no one who set out for the crystal mountain has ever returned. Between you and me, I think it is cursed," the innkeeper ended, looking Bob straight in the eye. "You'd better not go near it."

But, as is often the case in legends, bold young men rarely listen to the advice of elders. Therefore, it is probably not a big surprise that the next morning sun found Bob at the foot of the mountain that imprisoned Alice. He was walking around the huge clusters of crystal stretching up and across as far as the eye could see, not knowing where to start, when—behind a cliff—he saw a unique structure. It was a kind of door which, instead of a handle, had an amazing mechanism that was in constant motion. All its parts, gears, levers, chains, were made of crystal and shone so brightly that any prolonged look provoked dizziness. Bob was determined to figure out how to open it. As hours passed, he tinkered with the *Non-comprehensible and Recondite Apparatus* (NRA), to no avail. However, before the thought of giving in started to linger in his mind, he recalled the gift he has received recently. Bob tried to plug it into the mechanism, put the stone next to it; he tested all the ways of using it that came to his mind. In a last desperate attempt, he lifted the crystal to his eye, and… could not believe what he saw. No longer did he see something non-comprehensible or recondite. Instead, he saw a mechanism of *Dramatically Reduced Appearance* (DRA), the operation of which he worked out in no time. He flipped a few switches, turned a crank and, finally, was able to go inside.

The door immediately slammed shut behind him. Now he had no choice, but to finish his quest. He found himself in an enormous cave, probably kilometers wide and long. It was so large that he could only see its two giant walls, at the junction of which was the door by which he entered. However, he could not move freely through it, for it was divided by lower walls, forming a sort of square grid, with countless passages, bridges and walkways between them. At the horizon, somewhere inside the cave, he observed a Tower, which was bound to be Alice's castle. Roaming through this complicated net of corridors for half an hour, he began to realise that something was wrong. In fact, each and every square room he visited was exactly the same. To his horror, Bob realised that this was a labyrinth of *Virtually All the Same Spots* (VASS). Indeed, every room had exactly the same set of walkways connecting to it, except for rooms that were close to the cave's walls. He was even unable to get very far from the starting point, returning to it time and time again. The moment he understood he has a problem with navigating the labyrinth, he again turned to the amethyst for help. The very moment he took it out of his pocket he felt a slight pull towards one of the passages. When he turned his steps in that direction, the pulling did not stop, in fact, he could feel it even stronger, an incredible, almost magnetic force. The gemstone was showing him the way! After several minutes of walking through eerily identical rooms and passageways, the crystal stopped working for a moment. Bob realised that if he had continued in this direction, he would have made it far into the cavern, away from both its walls, but this was not the direction in which the tower was visible. Shortly after, the pull started anew, but this time the pattern of corridor choices was different. Again, after the adventurer made it through two dozen corridors, the pulling faded. Although this time the crystal became inactive for good, Bob already knew what to do, as he realised that the stone has shown him two vectors, the linear combination of which could lead him straight to the castle.

Three hours later he was already standing at the gates of Alice's palace. He got struck by what he saw inside. In the main hall, there was Alice, all her servants and a few of the townsfolk. The legend was true. They all were there, panic and distress in their eyes, all silhouettes ready to flee, chaos unraveling, a scene of fear and fright. But nobody was moving. They all have been turned in crystal. Bob walked over to where Alice was standing. He recognised her at once, for she exuded the aura of a true princess. Her hands were cold, like glass, seemed non-human. In front of her, he found a box, and in it, a small vial containing some purple liquid. The card fixed to its side read: "You, who has come all the way here, do not rejoice too early, for the task that remains to you is the most difficult. We shall play a game. In this room, not every statue was once a human. It is your role to *separate* the people who are *Numbed but Truly Alive* (NTA) from the statues which keep *Nothing but Treacherous Appearances* (NTA). This potion can revive the people, but you cannot waste a drop on a statue. Determine the answer without guessing, or perish and become a crystal yourself." Bob reached for the magic amethyst now almost reflexively. But nothing happened. There was no pull, and all the statues seen through the colored stone looked the same as before. The young man did not lose hope, but the passing hours of futile efforts made him more and more tired. He sat down by the edge of the chamber, wanting to gather his strength, maybe take a nap for a while. Could it be that the last riddle was going to overwhelm him? Just as Bob was about to fall asleep, he realised he felt some sort of gentle pulsing in his hand. Almost like a faint, barely noticeable heartbeat. It was the crystal again, giving him a clue. Bob understood what he has to do momentarily. Bob touched the necks of all the statues in turn, and where he felt a gentle, slow pulse, he used the potion. When he had used up the last drop, suddenly, at once, all the statues trembled. In the blink of an eye, everything changed, for he was now surrounded not by statues but by real, living people. It took a while for everyone to understand the situation they were in. Bob recounted the story of his rescue expedition to Princess Alice, who listened with keen attention and asked him repeatedly for more details. Alice and her subjects was excited and greatly exulted at being freed from the crystal prison. Bob was unanimously hailed a hero and hosted him in the castle with honors worthy of a king. There was no end to the joy, feasting and revelry.

They live happily ever since. Princess Alice rules her kingdom justly, providing a prosperous life for her subjects. Bob, having spent a few weeks at the court, has returned to his life of an adventurer on the move. They do keep in touch though. To this day, they still send encrypted messages to each other. They must have exchanged public keys. Who knows, maybe one day their paths will be reunited?

Chapter 1
Overview

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.

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

[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.

can recognise 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 realising 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 §1.2, we need to introduce the models of computation that are of interest in this dissertation.

1.1 Models of computation

Automata over infinite alphabets

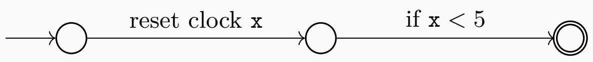
In the upcoming §§1.1.1 and 1.1.2, we present two models which both recognise words over infinite alphabets: $\Sigma \times \mathbb{Q}_{\geq 0}$ and $\Sigma \times \mathbb{A}$, where each input letter from a finite set Σ is paired with an element of an infinite set. In case of the first model—*timed 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 build upon standard finite automata. To deal with the extended alphabets, they are equipped with so-called *clocks* and *registers*, respectively.

1.1.1 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 automaton distinguishing timestamps

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 $\mathcal{A}$ with one clock $\mathtt{x}$ and three states.



It resets a clock $\mathtt{x}$ at the first symbol and then accepts if $\mathtt{x} < 5$ 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, 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) [17], and PRISM (probabilistic timed automata) [68]. The reachability problem is still a very active research area these days [1, 44, 51, 52, 55, 57], as are expressive generalisations thereof, such as the binary reachability problem [31, 41, 48, 67]. As a testimony to the model's importance, the authors of [2] received the 2016 Church Award [21] 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 [78], fault diagnosis [13], learning [91, 96], winning conditions in timed games [4, 14, 60], and in a notion of recognisability of timed languages [72].

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 on defining *determinisable* classes of timed automata, such as strongly non-zeno NTA [5], event-clock NTA [3], and NTA with integer-resets [89]. 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

[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.

interest to be able to decide whether a timed language presented as an NTA is actually recognisable by a DTA.

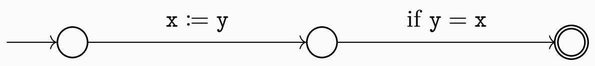
Another remarkable subclass of NTA is obtained by requiring the presence of just one clock (without epsilon transitions). The resulting class of NTA_1 is incomparable with DTA: For instance, NTA_1 are not closed under complement (unlike DTA), and there are very simple DTA languages that are not recognisable by any NTA_1. Nonetheless, NTA_1, like DTA, have decidable inclusion, equivalence, and universality problems [70, 79], although the complexity is non-primitive recursive [70, Corollary 4.2] (see also [80, Theorem 7.2] for an analogous lower bound for the satisfiability problem of metric temporal logic). Moreover, the nonemptiness problem for NTA_1 is NLogSpace-complete (vs. PSpace-complete for unrestricted NTA and DTA, already with two clocks [44]), and the binary reachability relation of NTA_1 can be computed as a formula of existential linear arithmetic of polynomial size, which is not the case in general [25].

1.1.2 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 [61]. 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 $w_1 = (a, \underline{0})(a, \underline{0})$ and $w_2 = (a, \underline{0})(a, \underline{1})$ can be distinguished by a register automaton $\mathcal{A}$ with one register $\mathtt{x}$ and three states:



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

Kaminski and Francez have shown, among other things, that nonemptiness is decidable [61, Theorem 1]. It was later realised that the problem is, in fact, PSpace-complete [40, Theorems 4.3 and Theorem 5.1]. The class of NRA recognisable languages is not closed under complementation [61, Proposition 5]; moreover, universality (and thus equivalence and inclusion) of NRA is undecidable [77, Theorem 5.1] (already for NRA with two registers [40, Theorem 5.4]).

Deterministic register automata

One way to regain decidability is to consider the deterministic register automata (DRA). This strict subclass of 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 [76]. A recent result completing

[3]In fact, even the inclusion problem $\mathcal{L}(A) \subseteq \mathcal{L}(B)$ with A an NRA and B a DRA is decidable.

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 NRA recognisable [64].

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 $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$ is decidable when $\mathcal{A}$ is an NRA and $\mathcal{B}$ is an NRA with one register [61, Appendix A], albeit the complexity is non-primitive recursive in this case [40, 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 $\mathbb{A}$ given as a parameter. In this setting, the original variant of register automata is obtained by choosing $\mathbb{A}$ 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.

1.1.3 Automata with counters (VASS)

Vector addition systems [62] (VAS) are a widely accepted model of concurrency equivalent to Petri nets. Another equivalent model, called *vector addition systems with states* (VASS) [58], 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 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 [73] was published in 1981. However, the resulting Ackermann-completeness of the problem (dropping to PSpace-completeness for 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 VASS_2.

Described in the dissertation is a *pumping method* for runs of 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 VASS, as even the core of the seminal decision procedure for the reachability problem in VASS by Mayr and Kosaraju [66, 74] is fundamentally based on pumping. The decision procedure they propose essentially decomposes a VASS into a finite number of more structured 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 VASS of bounded dimension. The PSPACE upper bound for the reachability problem in 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 (NL) in case when the transition effects are represented in unary [43]. Un-pumping is also used in [19] to provide a quadratic bound on the length of the shortest run for VASS_1, also known as one counter automata without zero tests, and for unrestricted one counter automata. See also [6, 71] for pumping techniques in one counter automata.

1.2 Contributions grouped by problems

Here, we summarise the main results found in the dissertation. Each of following §§1.2.1 to 1.2.4 first sketches the problem and the state of the art on it, and then presents the newly proven theorems.

1.2.1 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 NRA/NTA, and the goal is to provide a deterministic device realising it. Before we discuss it, we need to introduce the notions of ω-regular languages and synthesis games.

ω-regular languages

For finite automata over alphabet Σ, there are many ways of adapting them to recognise *infinite words* (from Σ^ω) instead of finite ones (from Σ^*). Some of these approaches only modify semantics, while others introduce subtle alterations of the syntax, too. One may consider DRA or NRA with updated acceptance conditions, i.a., Büchi, Muller, or parity condition. Most[4] of these variants have the same set of languages they can express—this way, the class of *ω-regular* languages arises.

[4] All classes but DRA equipped with Büchi condition, which can only express a strict subset of ω-regular languages.

Synthesis games

One of the areas in which ω-regular languages find their use are *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 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 *finite memory strategy* in such a game is a deterministic automaton with additional output[5] letter written on every transition. It reads the opponent's actions one by one and responds to them synchronously, with current player's actions.

Synthesis problem

In their famous result [15], Büchi and Landweber have shown that one can

- decide the winner of such games,
- compute (synthesize) a finite-memory strategy for the winner.

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 *timed synthesis games* and *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.

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] χ_i from [$\mathbb{Q}_{\geq 0}/\mathbb{A}$]. 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, π belongs to her winning set W, which is specified as the language of a [NTA/NRA] (possibly with ε-transitions).

Analogously to original synthesis games, finite-memory strategies in above game correspond to [timed/register] automata with output.

Timed and register synthesis problems

The *[timed/register] synthesis problem* is to compute, if it exists, a winning controller for Bob —[DTA/DRA] with output—which ensures that every play π conformant to that controller is winning for Bob. We also consider two more restricted variants of it. For a fixed number of [clocks/registers] $k \in \mathbb{N}$, the *[k-clock timed/k-register] synthesis problem* asks—as before—about a controller for Bob, but this time using at most k [clocks/registers]. Finally, exclusively for the timed variant, we define a *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

[5]Other names by which such automaton with output is known include 'letter-to-letter transducer', and 'Mealy machine'.

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.

Theorem 5.1 synthesis of DRA_k

For every fixed $k \in \mathbb{N}$, the k-register synthesis problem is computable.

Theorem 6.1 synthesis of $\mathsf{DTA}_{k,m}$

For every fixed $k, m \in \mathbb{N}$, the k-clock m-constrained timed synthesis problem is computable.

Theorem 6.2 synthesis of DTA_k

For every fixed $k \in \mathbb{N}$, the k-clock timed synthesis problem is computable.

The main part of our proof is the reduction of these games to finite-state games with an ω-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 [15].

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. [42] for a similar observation).

When compared with our setup, variant of register synthesis problem considered by Khalimov et al. [63] (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 8.4 synthesis of DRA und.

The register synthesis decision problem is undecidable, and this holds already when Alice's winning condition is an NRA_2 language.

and of timed synthesis, too:

Theorem 8.2 synthesis of DTA und.

The timed synthesis decision problem is undecidable, and this holds already when Alice's winning condition is an NTA_2 language.

Moreover, as in [63] the transducer is limited to use only atoms stored in registers, in context of [k-clock/k-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 [63], while additionally allowing other atom domains.

1.2.2 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 $S \cap M = \varnothing$.

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 [85]), 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 [34]), 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 [36, 81] (later generalised to context-free languages [37] and finite trees [54]), locally and locally threshold testable languages [82], first-order logic definable languages [84] (generalised to some fixed levels of the first-order hierarchy [83]). 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 [59, 65, 90]. Nonetheless, there are positive decidability results for separability problems on several infinite-state models, such as Petri nets [24], Parikh automata [22], one-counter automata [32], higher-order and collapsible pushdown automata [30, 56], 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 $\mathcal{A}$ and $\mathcal{B}$ with disjoint languages, be it timed or register ones, whether there exists a deterministic automaton $\mathcal{S}$ such that $\mathcal{L}(\mathcal{S})$ separates $\mathcal{L}(\mathcal{A})$ from $\mathcal{L}(\mathcal{B})$. 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 $\mathcal{A}, \mathcal{B} \in \mathsf{NTA}$ can be separated by a deterministic automaton $\mathcal{S} \in \mathsf{DTA}$ 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 $\mathcal{S}$ can measure the time up to arbitrary fixed number of time units.
- Finally, the *k-register deterministic separability* for register automata asks whether $\mathcal{A}, \mathcal{B} \in \mathsf{NRA}$ can be separated by a deterministic automaton $\mathcal{S} \in \mathsf{DRA}$ 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 $\mathcal{A}$ as recognising a set of good behaviours which we want to preserve and $\mathcal{B}$ 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 §1.2.1. We provide analogous results for both timed and register automata:

Theorem 4.1 separability by DRA_k

For $k \in \mathbb{N}$, the k-register deterministic separability problem for NRA is decidable.

Theorem 4.3 separability by $\mathsf{DTA}_{k,m}$

For $k, m \in \mathbb{N}$, the k-clock m-constrained separability problem for NTA is decidable.

Theorem 4.2 separability by DTA_k

For $k \in \mathbb{N}$, the k-clock deterministic separability problem for NTA are 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 [46, 93] (also cf. §1.2.3). This is a rare circumstance, which is shared with languages recognised by one-counter nets [32], and conjectured to be the case for the full class of Petri net languages[6].

Similar situation holds also for register automata, as we outline in the next section §1.2.3.

1.2.3 Deterministic membership problem

Here, we consider an instance of the question Q3:

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

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

Timed automata

The DTA *membership problem* asks, given an NTA, whether there exists a DTA recognising the same language. There are two natural variants of this problem, which are obtained by restricting the resources available to the sought 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 DTA_k is a variant of the problem above where the DTA is required to have at most k clocks.
- The $\mathsf{DTA}_{k,m}$ *membership problem* asks for the automaton with at most k clocks and constants bounded by m in absolute value.

Notice that we do not bound the number of control locations of the DTA, which makes the problem non-trivial. (Indeed, there are finitely many DTA with a bounded number of clocks, control locations, and maximal constant.)

Since untimed regular languages are deterministic, the DTA_k membership problem can be seen as a quantitative generalisation of the regularity problem. For instance, the 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 [88, Theorem 6.6.6], labelled Petri nets under reachability semantics [95], Parikh automata [16], etc. One way to obtain decidability is to either restrict the input model to be deterministic (e.g., [7, 94, 95]), or to consider more refined notions of equivalence, such as bisimulation (e.g., [53]).

[6]All these classes of languages have a decidable disjointness problem, however regular separability is not always decidable in this case [92].

This negative situation is generally confirmed for timed automata. For every number of clocks $k \in \mathbb{N}$ and maximal constant m, the DTA, DTA_k, and $\mathsf{DTA}_{k,m}$ membership problems are known to be undecidable when the input NTA has ≥ 2 clocks, and for 1-clock NTA with epsilon transitions [46, 93]. To the best of our knowledge, the deterministic membership problem was not studied before when the input automaton is NTA_1 without epsilon transitions.

Register automata

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

- The DRA_k *membership problem* asks, given an NRA, whether there exists a DRA with k registers recognising the same language.
- The 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 [46, 93], and we extend these results to register automata.

Upper bounds

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

Theorem 7.28 DTA_k membership

For every fixed $k \in \mathbb{N}$, the DTA_k membership problem is decidable for NTA_1 languages.

Theorem 7.29 $\mathsf{DTA}_{k,m}$ membership

For every fixed $k, m \in \mathbb{N}$, the $\mathsf{DTA}_{k,m}$ membership problem is decidable for 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 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 (cf. lemma 7.30). This provides a decision procedure since there are finitely many different 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 NTA_k. These automata are required to reset at least one clock at every transition and are thus of expressive power intermediate between NTA_{k-1} and NTA_k. Always resetting NTA_2 are strictly more expressive than 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 NTA_2 but by no NTA_1. Despite their increased expressive power, always resetting NTA_2 still have a decidable universality problem (the well-quasi order approach of [79] goes through), which is not the case for NTA_2. Thanks to this restricted form, we are able to provide in lemma 7.30 an elegant characterisation of those NTA_1 languages which are recognised by an always resetting DTA_k.

We prove a result analogous to theorem 7.28 in the setting of register automata.

Theorem 7.1 DRA_k membership

For every fixed $k \in \mathbb{N}$, the DRA_k membership problem is decidable for NRA_1 languages.

Thanks to the effective elimination of ε-transition rules from NRA_1 (cf. lemma 2.22), the decidability result above also holds for data languages presented as NRA_1 with ε-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 8.1 DTA and $\mathsf{DTA}_{\bullet,m}$ membership und.

The DTA and $\mathsf{DTA}_{\bullet,m}$ $(m > 0)$ membership problems are undecidable for NTA_1 without epsilon transitions.

Theorem 8.1 is shown by improving on an analogous result from [46, Theorem 1] for NTA_2. We obtain a similar undecidability result in the setting of register automata:

Theorem 8.3 DRA membership und.

The DRA membership problem is undecidable for NRA_1.

The following lower bounds further refine the analysis from [46] in the case of a fixed number of clocks of a deterministic acceptor.

Theorem 7.48 undecidability and hardness for DTA_k membership

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

1. undecidable for NTA_2,
2. undecidable for $\mathsf{NTA}_1^{\varepsilon}$ (NTA with epsilon transitions),
3. HyperAckermann-hard for 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 NRA_2 or an NRA_1 with guessing[7]. In the decidable case of an NRA_1 input, the problem is nonetheless not primitive recursive.

[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*.

Theorem 7.17 undecidability and hardness for DRA_k membership

Fix a $k \geq 0$. The DRA_k membership problem is:

1. undecidable for NRA_2,
2. undecidable for NRA_1^g (NRA_1 with guessing), and
3. not primitive recursive (Ackermann-hard) for NRA_1.

1.2.4 Pumping technique for VASS of dimension two

The pumping techniques mentioned in §1.1.3 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. [23, 35, 39]).

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 π of a VASS_2 starting and ending with both counters set to zero,

- either π 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 π 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 π are all bounded polynomially in M and exponentially in n.

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 9.7 thin/thick dichotomy

There is a polynomial p such that every $(0, 0)$-run in a VASS_2 V is either $p(nM)^n$-thin or $p(nM)^n$-thick.

The dichotomy immediately entails a pumping theorem for VASS_2.

Theorem 9.21 pumping runs of VASS_2

There is a polynomial p such that every $(0, 0)$-run τ in a 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 τ_0 and τ_k are also bounded by $p(nM)^n$.

Intuitively, adapted pumping scheme of VASS_1 is used in case of thin runs, whereas for thick runs we 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 PSPACE-membership of the reachability problem.

Theorem 9.22 exponential run property

There is a polynomial p such that for every $(0,0)$-run τ in a VASS_2, there is a $(0,0)$-run of length bounded by $p(nM)^n$ with the same source and target as τ.

1.3 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* [29] (ICALP 2020)
   Authors: Lorenzo Clemente, Sławomir Lasota and P.
   ▶ theorems 6.1, 6.2 and 8.2 (timed synthesis)
   ▶ theorems 4.2 and 4.3 (DTA separability)
2. *Determinisability of One-Clock Timed Automata* [26] (CONCUR 2020)
   Authors: Lorenzo Clemente, Sławomir Lasota and P.
   ▶ theorems 7.28, 7.29 and 7.48 (DTA membership)
3. *Determinisability of register and timed automata* (submitted to LMCS)
   Authors: Lorenzo Clemente, Sławomir Lasota and P.
   ▶ theorems 7.1, 7.17, 7.28, 7.29 and 7.48 (DTA and DRA membership)
4. *New Pumping Technique for 2-Dimensional VASS* [33] (MFCS 2019)
   Authors: Wojciech Czerwinski, Slawomir Lasota, Christof Löding and P.
   ▶ theorems 9.21 and 9.22 (pumping technique for VASS_2)
5. Previously unpublished results
   Authors: Lorenzo Clemente, Sławomir Lasota and P.
   ▶ theorems 5.1 and 8.4 (register synthesis)
   ▶ theorem 4.1 (DRA separability)

CHAPTER 2

PRELIMINARIES

Here we introduce concepts essential for this dissertation. We first discuss the underlying notions of transition systems and sets with atoms. The main purpose of this chapter is to present three computational models with *infinite configuration space* and their associated computational problems, which are of interest to us in the sequel.

This chapter has a complementary character to §§1.1 and 1.2. We present here precise definitions of notions which were only briefly sketched in §1, as the main focus was on discussing the current state of research in their field.

BASIC NOTATIONS

Sets of numbers We use standard symbols $\mathbb{R}, \mathbb{Q}, \mathbb{Z}, \mathbb{N}$ for the sets of reals, rationals, integers, and non-negative integers, respectively. Whenever convenient, we use subscripts to specify subsets, e.g., $\mathbb{R}_{\geq 0}, \mathbb{Q}_{\geq 0}$ for non-negative reals and rationals.

Notational simplifications and shortcuts When speaking of composite objects, we use the symbol '_' to refer to unused components. For example, when defining a projection $\phi : A \times B \times C \to A \times C$, we write $f(a, \_, c) = (a, c)$. Given a set $S \subseteq A \times B \times C$ and two elements $a \in A, c \in C$, when writing 'fix an element of S of shape $(a, \_, c)$, we mean arbitrary $(a', b', c') \in S$ such that $a' = a$ and $c' = c$. When speaking jointly about elements of some sequence $a_1, a_2, a_3, \ldots$, we write $a_\bullet$.

2.1 Models of computation with infinite space of configurations

The automata theory studies a wide variety of models of computation, ranging from very simple (thus less expressive) ones, to more powerful machines, which are in turn harder to analyze. On one end of the spectrum there are finite automata—its founding concept. They are long known, thus well studied and understood, but can model only very simple processes. At the other end, there are Turing machines, being able to perform any[1] computation. There is, however, a high price one has to pay for their versatility—all nontrivial problems about languages they recognise are undecidable [86].

[1]Assuming the validity of the Church-Turing Thesis.

The space between these vastly different models did not remain empty and is populated by intermediate models which emerged as a compromise between weaker expressiveness and difficulty of theoretical analysis. Classical examples of such models include pushdown automata, one counter automata, linearly bounded Turing machines or vector addition systems with states (VASS). All these models feature an infinite configuration space, as Turing machines do, but still have some non-trivial properties decidable, e.g., language membership or emptiness problem.

All models of computation mentioned so far share yet another common quality—they are acceptors of languages over some finite alphabet Σ, or they can be easily extended to become such acceptors. However, this is not something to take for granted, as there exist models going beyond this regime, for instance timed automata, hybrid automata, tree-walking automata, register automata, and orbit-finite automata. These devices operate on richer data structures: timed words, trees, data words, and words over an orbit-finite alphabet, correspondingly. Such upgrade is twofold beneficial:

- the expressive power is increased, while the machines stay relatively 'simple'—the input structures are read sequentially and only once, unlike in Turing machines,
- resulting models are by design well-suited for specific practical applications in analysis of real-life systems—e.g., timed automata shine at modelling real-time reactive systems, while tree-walking automata are geared well towards defining and checking properties of XML documents.

In this dissertation, we focus primarily on three models of computation: register automata, timed automata and vector addition systems. They all feature an infinite space of configurations; this section's purpose is to introduce all three. We begin by giving the shared terminology that will prove useful when defining the classes of models in §§2.4 to 2.6.

2.2 Labelled transition systems (LTS)

Transition systems (TS) are a universal tool that can be used to specify the semantics of all three models of our interest. They are (possibly infinite) directed graphs, whose vertices are called configurations, their edges forming a transition relation. More formally, a transition system $\mathcal{T} = (C, C_{\mathsf{ini}} \rightarrow)$ is a tuple, where:

- C is the (possibly infinite) set of *configurations*,
- $\rightarrow \subseteq C \times C$ is the transition relation between configurations,
- $C_{\mathsf{ini}} \subseteq C$ is a subset of initial configurations.

An element of transition relation $(c, c') \in \rightarrow$ is also written as $c \rightarrow c'$.

Runs of TS

A *run* π in $\mathcal{T} \in \text{TS}$, starting in configuration c_0, ending in configuration c_n is a finite walk in the graph $(C, \rightarrow)$ of the form

$$\pi = (c_0, c_1)(c_1, c_2) \ldots (c_{n-1}, c_n) \in \rightarrow^*$$

also written as

$$\pi = c_0 \rightarrow c_1 \rightarrow \ldots \rightarrow c_n.$$

If there exists a run π from c to c', we write $c \rightarrow^* c'$. $\mathsf{Runs}(\mathcal{T}; c, c')$ is a set of all runs starting in configuration c and ending in c'. Additionally, we define:

$$\begin{aligned} \mathsf{Runs}(\mathcal{T}; c) &= \bigcup_{c' \in C} \mathsf{Runs}(\mathcal{T}; c, c') \\ \mathsf{Runs}(\mathcal{T}) &= \bigcup_{c \in C_{\mathsf{ini}}} \mathsf{Runs}(\mathcal{T}; c). \end{aligned}$$

In words, $\mathsf{Runs}(\mathcal{T})$ is the set of all runs of $\mathcal{T}$ which start in any initial configuration. A configuration c' is *reachable* if there exists $c \in C_{\mathsf{ini}}$ such that $c \rightarrow^* c'$.

Labelled transition systems

Transition systems also appear in a labelled variety, called *labelled transition systems* (LTS). This variant is suitable for defining semantics of models of computation which read some input word. Intuitively, an LTS is identical to a finite automaton, where the requirement of finiteness of the set of states has been lifted. More formally, a labelled transition system $\mathcal{T} = (\Gamma, C, C_{\mathsf{ini}}, C_{\mathsf{fin}}, \rightarrow)$ is a tuple, where:

- Γ is the set of *transition labels*,
- C is the (possibly infinite) set of *configurations*,
- C_{ini} and C_{fin} are subsets of C called *initial* and *final* configurations, respectively,
- $\rightarrow \subseteq C \times \Gamma \times C$ is the transition relation between configurations.

Similarly to TS, we write $c \xrightarrow{\gamma} c'$ to denote a transition $(c, \gamma, c') \in \rightarrow$.

An LTS is *deterministic*, if $|C_{\mathsf{ini}}| = 1$, and for every configuration $c \in C$ and label $\gamma \in \Gamma$ there exists at most one configuration $c' \in C$ such that $c \xrightarrow{\gamma} c'$. It is *total*, when for every $c \in C$ and $\gamma \in \Gamma$ there exists at least one configuration $c' \in C$ such that $c \xrightarrow{\gamma} c'$.

The class of deterministic LTS is denoted DLTS. For $\mathcal{T} \in \text{LTS}$ with set of labels Γ, we say that $\mathcal{T}$ is *over* Γ. By $\text{LTS}(\Gamma)$ (or $\text{DLTS}(\Gamma)$) we mean the class of all LTS (DLTS, respectively) over Γ.

Runs of LTS

A *run* π in $\mathcal{T}$, starting in configuration c_0, ending in configuration c_n and labelled by a word $w = \gamma_1\gamma_2 \dots \gamma_n \in \Gamma^*$ (or "over" w), is a finite labelled walk in the graph $(C, \rightarrow)$ of the form

$$\pi = (c_0, \gamma_1, c_1)(c_1, \gamma_2, c_2) \dots (c_{n-1}, \gamma_n, c_n) \in \rightarrow^*$$

also written as

$$\pi = c_0 \xrightarrow{\gamma_1} c_1 \xrightarrow{\gamma_2} \dots \xrightarrow{\gamma_n} c_n.$$

For such run, we define $\mathsf{input}(\pi) = w$. We say that π is *accepting* if its last configuration is final ($c_n \in C_{\mathsf{fin}}$). If there exists a run π from c to c' labelled with w, we write $c \xrightarrow{w}^* c'$; if some such w exists, we write $c \rightarrow^* c'$. $\mathsf{Runs}(\mathcal{T}; c, c')$ is a set of all runs starting in configuration c and ending in c'. Additionally, we define:

$$\mathsf{Runs}(\mathcal{T}; c) = \bigcup_{c' \in C} \mathsf{Runs}(\mathcal{T}; c, c')$$
$$\mathsf{Runs}(\mathcal{T}) = \bigcup_{c \in C_{\mathsf{ini}}} \mathsf{Runs}(\mathcal{T}; c).$$

In words, $\mathsf{Runs}(\mathcal{T})$ is the set of all runs of $\mathcal{T}$ which start in any initial configuration. A configuration c' is *reachable* if there exists $c \in C_{\mathsf{ini}}$ such that $c \rightarrow^* c'$. Note that in case of deterministic LTS, for every $w \in \Gamma^*$ there exists at most one run π over w, while for a total LTS at least one run needs to exist (not necessarily accepting).

Language of LTS

The language $\mathcal{L}(\mathcal{T}; c) \subseteq \Gamma^*$ of a configuration $c \in C$ is defined in a natural way as the set of possible labels of runs from c to some final state:

$$\mathcal{L}(\mathcal{T}; c) = \left\{\mathsf{input}(\pi) \,\middle|\, \exists_{c' \in C_{\mathsf{fin}}} \,.\, \pi \in \mathsf{Runs}(\mathcal{T}; c, c')\right\}$$

The language of LTS $\mathcal{T}$ is a sum of languages of all its initial configurations:

$$\mathcal{L}(\mathcal{T}) = \bigcup_{c \in C_{\mathsf{ini}}} \mathcal{L}(\mathcal{T}; c)$$

Infinite runs

An *ω-run* π in $\mathcal{T}$, starting in configuration c_0 and labelled with an infinite word $w = \gamma_1\gamma_2\gamma_3 \dots \in \Gamma^\omega$, is an infinite labelled walk in the graph $(C, \rightarrow)$ of the form

$$\pi = (c_0, \gamma_1, c_1)(c_1, \gamma_2, c_2)(c_2, \gamma_3, c_3) \dots \in \rightarrow^\omega,$$

written also as

$$\pi = c_0 \xrightarrow{\gamma_1} c_1 \xrightarrow{\gamma_2} c_2 \xrightarrow{\gamma_3} \dots.$$

The set of all ω-runs of $\mathcal{T}$ starting in c_0 is denoted as $\mathsf{Runs}^\omega(\mathcal{T}; c_0)$, and—quite naturally—we define $\mathsf{Runs}^\omega(\mathcal{T}) = \bigcup_{c \in C_{\mathsf{ini}}} \mathsf{Runs}^\omega(\mathcal{T}; c)$. For a run π as above, we define $\mathsf{Inf}(\pi)$ to be the set of configurations visited by π infinitely many times

$$\mathsf{Inf}(\pi) = \left\{c \in C \,\middle|\, \big|\{i \in \mathbb{N} \mid c_i = c\}\big| = \aleph_0\right\}.$$

Languages of ω-words

Infinite words, a.k.a. ω-words, are infinite sequences of letters from some alphabet. The set of all ω-words over the alphabet Σ is denoted as Σ^ω. For further application in this work, we will present two definitions of the language of w-words (or ω-language) which can be associated with $\mathcal{T}$. When there is no danger of confusion, we sometimes drop the prefix 'ω-' and speak of words, languages, runs, etc.

Language Reach(L)

For a language $L \subseteq \Gamma^*$ of finite words over the alphabet Γ, we define the *reachability language* Reach(L) as the set of ω-words, whose some prefix belongs to L. In case of LTS $\mathcal{T}$, Reach$(\mathcal{L}(\mathcal{T})) \subseteq \Gamma^\omega$ is the set of labels of runs of $\mathcal{T}$, which visit some final state at least once. We will denote that ω-language as Reach$(\mathcal{T})$.

Büchi acceptance condition

We obtain a more general family of ω-languages of LTS as a result of using so-called 'Büchi winning condition'. In this case, we say that an ω-run π is Büchi-accepting, whenever it visits final states infinitely many times

$$\mathsf{Inf}(\pi) \cap C_{\mathsf{fin}} \neq \varnothing.$$

The careful reader may wonder whether the choice of this variant of the acceptance condition is not too restrictive. In the theory of ω-regular languages, there are many other ways of defining a language of infinite words, for instance by making use of Muller or parity conditions. More worryingly, Büchi's condition applied to deterministic systems is not sufficient to define the full class of omega-regular languages. In this dissertation, however, in conjunction with Büchi's condition, we will consider only nondeterministic models of computation. Moreover, the condition's simplicity (no changes to the syntax of the automaton, only modified semantics) will be a desirable feature. Places where it is used in the following chapters can be easily adapted for the use of other conditions.

Language $\mathcal{L}^\omega(\mathcal{T})$

For a configuration $c \in C$, we define its *ω-language* $\mathcal{L}^\omega(\mathcal{T}; c) \subseteq \Gamma^\omega$ as the set of labels of runs starting in c and satisfying Büchi acceptance condition:

$$\mathcal{L}^\omega(\mathcal{T}; c) = \{\mathsf{input}(\pi) \mid \pi \in \mathsf{Runs}^\omega(\mathcal{T}) \wedge \ \pi \text{ is Büchi-accepting}\}.$$

As in the finite case, we define

$$\mathcal{L}^\omega(\mathcal{T}) = \bigcup_{c \in C_{\mathsf{ini}}} \mathcal{L}^\omega(\mathcal{T}; c).$$

Lemma 2.1.

For any LTS $\mathcal{T}$, there exists an LTS $\mathcal{T}'$ such that Reach$(\mathcal{T}) = \mathcal{L}^\omega(\mathcal{T}')$.

Proof of lemma 2.1.

Fix an LTS $\mathcal{T}$. Construct $\mathcal{T}' = (\Gamma, C \cup \{\mathsf{Acc}\}, C_{\mathsf{ini}}, \{\mathsf{Acc}\}, \to')$ by adding a new configuration Acc, which becomes the only accepting one in $\mathcal{T}'$. The new transition relation $\to'$ extends $\to$ with the transitions

$$c \xrightarrow{\gamma} \mathsf{Acc}, \quad \mathsf{Acc} \xrightarrow{\gamma} \mathsf{Acc}$$

for every $c \in C_{\mathsf{fin}}$ and $\gamma \in \Gamma$.

For the left to right inclusion, fix $w \in \mathcal{L}(\mathcal{T})$ and $v = v_0 v_1 v_2 \cdots \in \Gamma^\omega$. Word wv is Büchi-accepted by $\mathcal{T}'$. Indeed, let π be a run from $\mathsf{Runs}(\mathcal{T})$ which accepts w and ends in configuration $c \in C_{\mathsf{fin}}$. We construct π' by extending π with the transition $c \xrightarrow{v_0} \mathsf{Acc}$ and then using the self-loops $\mathsf{Acc} \xrightarrow{v_i} \mathsf{Acc}$ with letters $v_1, v_2, \ldots$. It is easy to see that the constructed run π' accepts wv. For the right to left inclusion, observe that if word w belongs to $\mathcal{L}^\omega(\mathcal{T}')$, then the corresponding ω-run π must have visited some state $c \in C_{\mathsf{fin}}$ before going to Acc. The prefix of π ending at the last occurrence of c is an accepting run of $\mathcal{T}$ and therefore $w \in \textsc{Reach}(\mathcal{T})$. □

The above lemma shows that languages of LCM defined with the Büchi winning condition are at least as expressive as the reachability languages. The following example shows that the converse statement is not true.

Example 2.2. ω-language not of the form $\textsc{Reach}(\bullet)$

We define a language of words $w \in \{a, b\}^\omega$ which contain infinitely many letters a. Let $\mathcal{T} = (\{a, b\}, \{\mathsf{s}_a, \mathsf{s}_b\}, \{\mathsf{s}_b\}, \{\mathsf{s}_a\}, \to)$, where the transition relation contains the elements

$$c \xrightarrow{a} \mathsf{s}_a, \quad c \xrightarrow{b} \mathsf{s}_b,$$

where $c \in \{\mathsf{s}_a, \mathsf{s}_b\}$. Language $L = \mathcal{L}^\omega(\mathcal{T})$ is not of the form $\textsc{Reach}(\mathcal{T}')$ for any LTS $\mathcal{T}'$. Assume otherwise, and fix any $\mathcal{T}'$ such that $L = \textsc{Reach}(\mathcal{T}')$. Consider a word $a^\omega \in L$. It has to belong to $\textsc{Reach}(\mathcal{T}')$, therefore for some $n \in \mathbb{N}$ the prefix a^n belongs to $\mathcal{L}(\mathcal{T}')$. This in turn implies that $a^n b^\omega \in \textsc{Reach}(\mathcal{T}')$, yielding a contradiction.

LTS WITH OUTPUT

An LTS *with an output* $\mathcal{T}$ is a deterministic and total LTS additionally equipped with an *output alphabet* Υ, with every transition labeled by an *output label* $\upsilon \in \Upsilon$. Formally, is a tuple $\mathcal{T} = (\Gamma, \Upsilon, C, C_{\mathsf{ini}}, \delta)$, such that

- $\mathcal{T}' = (\Gamma, C, C_{\mathsf{ini}}, \varnothing, \delta')$ is a *deterministic and total* LTS, where δ' is $\{(c_1, \gamma, c_2) \mid \exists_{\upsilon \in \Upsilon}(c_1, \gamma, \upsilon, c_2)\}$,
- in δ, output labels are functionally determined by other components, i.e., for any configurations $c_1, c_2 \in C$ and label $\gamma \in \Gamma$ there exists exactly one letter $\upsilon \in \Upsilon$ such that $(c_1, \gamma, \upsilon, c_2) \in \delta$.

Note that in LTS with output, the set of final configurations is not important and assumed to be empty.

Notions of finite or infinite runs defined for $\mathcal{T}' \in$ LTS naturally transfer to LTS with output $\mathcal{T}$. A transition $(c_0, \gamma, \upsilon, c_1) \in \delta$ is also written as

$$c_0 \xrightarrow{\gamma/\upsilon} c_1.$$

For a run

$$\pi = c_0 \xrightarrow{\gamma_1/\upsilon_1} c_1 \xrightarrow{\gamma_2/\upsilon_2} c_2 \xrightarrow{\gamma_3/\upsilon_3} \dots$$

by analogy with $\mathsf{input}(\pi) = \gamma_1\gamma_2\gamma_3\dots$, we define $\mathsf{output}(\pi) = \upsilon_1\upsilon_2\upsilon_3$, both for finite and infinite runs.

With each word w, finite or infinite, we associate a word $\mathcal{T}(w) \coloneqq \mathsf{output}(\pi)$, where π is the unique run of $\mathcal{T}$ over w. In case when $\mathcal{T}' \in$ DFA, we call $\mathcal{T}$ *an automaton with output*[2].

Class of LTS with output over alphabets Γ (input) and Υ (output) is denoted as DLTS(Γ, Υ), similar notation being used for finite automata: DFA(Γ, Υ).

Similarities of models to be defined

In the upcoming subsections, in order to define semantics of the three models we plan to introduce, we will instantiate the general framework of TS/LTS by plugging in as configurations and labels some sets of more definite shape. It is instructive to look at them now, even before we are ready to formally define their corresponding transition relations $\to_\bullet$, as similarities between the models will already become apparent.

For comparison, let us first look at LTS corresponding to some finite automaton $\mathcal{A} = (\Sigma, L, L_{\mathsf{ini}}, L_{\mathsf{fin}}, \delta)$ with control locations L recognizing words over a finite alphabet Σ. As LTS generalise finite automata, the construction is straightforward:

0. LTS for a finite automaton

labels	$\Gamma_0 = \Sigma$
configurations	$C_0 = L$
transitions	$\to_0 = \delta$

Configurations and labels of LTS instantiated to define semantics of register automata, timed automata and VASS look as follows:

1. LTS for a k-register automaton

labels	$\Gamma_1 = \Sigma \times \mathbb{A}$
configurations	$C_1 = L \times (\mathbb{A} \cup \{\bot\})^{\mathtt{X}}$

2. LTS for a k-clock timed automaton

labels	$\Gamma_2 = \Sigma \times \mathbb{Q}_{\geq 0}$
configurations	$C_2 = L \times \mathbb{Q}_{\geq 0}^{\mathtt{X}} \times \mathbb{Q}_{\geq 0}$

3. TS for VASS of dimension k

configurations	$C_3 = L \times \mathbb{N}^k$

[2]This notion appears in the literature also under equivalent names ‘letter-to-letter transducer’ and ‘Mealy machine’.

where $\mathbb{A}$ is some countably infinite set of so called *atoms* (to be discussed in greater detail in §2.3), $\mathtt{X} = \{\mathtt{x}_1, \mathtt{x}_2, \ldots, \mathtt{x}_k\}$ is a k-element set of names, and $\mathbb{Q}_{\geq 0}, \mathbb{N}$ denote nonnegative rational numbers and natural numbers, correspondingly.

Observe that:

- all sets of configurations are infinite,
- as in finite automata, all labels in LTS feature a letter from a finite alphabet Σ, while all configurations have control location coming from a finite set L,
- configurations of all models feature a mapping from some k-element set (be it register names, clock names, or dimension numbers) to some infinite set,
- register and timed automata recognise words over *infinite alphabets* (Γ_1 and Γ_2).

Similarities between timed and register automata

As of now, one could notice a striking resemblance between register and timed automata in terms of the shape of their configurations and labels. It is not a coincidence, as both models share many properties (e.g., see [45] for a link concerning a class of language-related decision problems). Although several secondary but nonetheless not negligible technical differences between the models force us to treat timed automata separately[3], ultimately, the results for both models that we prove are remarkably similar.

2.3 Sets with atoms

As we have just signalled, register automata operate on words over an infinite alphabet $\Sigma \times \mathbb{A}$. Now, we will specify what exactly is the meaning of the set $\mathbb{A}$. This section's role is to present a part of the theory of *sets with atoms* with which the register automata are directly and inextricably linked. Due to the similarities between the models a great portion of it will prove useful when defining the timed automata as well.

2.3.1 Atoms

Register automata are parametrised by their data domain—some relational structure $\mathbb{A}$ over a finite signature, and with a countably infinite universe. We will call elements of $\mathbb{A}$ *atoms* henceforth. As there is no danger of confusion, we will use the symbol $\mathbb{A}$ both to denote the structure itself and its universe.

The simplest type of atoms—so called *equality atoms* $\mathbb{A} = \langle \mathbb{N}, = \rangle$ contains in its signature the equality symbol only. For this reason, any other countably infinite set would be indistinguishable from $\mathbb{N}$. To indicate that we are treating numbers as atoms—and, therefore, do not care about their order—we use the underline, e.g.: $\underline{1}, \underline{5}, \underline{7}$.

Other kinds of atoms include:

A. *densely ordered atoms $\langle \mathbb{Q}, \leq \rangle$* where $\mathbb{Q}$ is the set of rational numbers and '$\leq$' is the natural order),

[3]As is made evident by the different degrees of intricacy of proofs between §§5 and 6 or between proofs in §8.

B. *timed atoms* $\langle \mathbb{Q}, \leq, +1 \rangle$ which extend densely ordered atoms by the increment relation. A register automaton operating on timed atoms closely resembles a timed automaton.

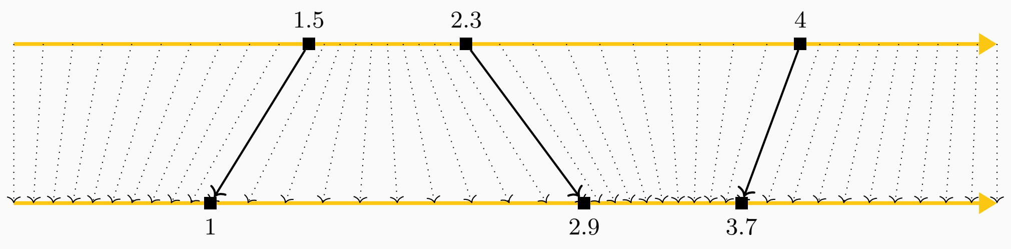
C. $\langle \mathbb{N}^2, =_1, = \rangle$: pairs of elements which can be tested for equality of the first coordinate $(=_1)$ and of both coordinates $(=)$.

Homogeneous atoms

So-called homogeneity is a key property that enables algorithmic manipulation of computational models equipped with atoms. It allows for a finite representation of certain objects associated with atoms, so that they can constitute input or output of for algorithms (cf. claim 2.15). Formally, a structure $\mathbb{A}$ is *homogeneous* if every isomorphism between some two of its finitely generated substructures can be extended to an automorphism of $\mathbb{A}$. This dissertation focuses solely on homogeneous atoms.

Example 2.3. (standard) example of a homogeneous structure

Densely ordered atoms $\mathbb{A} = \langle \mathbb{Q}, \leq \rangle$ are homogeneous. Indeed, every finite partial automorphism of $\mathbb{A}$ (i.e., a function preserving $\leq$) can be extended to a full automorphism of $\mathbb{A}$. Consider a partial function $f = \{1.5 \mapsto 1, 2.3 \mapsto 2.9, 4 \mapsto 3.7\} \in \mathbb{A} \times \mathbb{A}$. Picture below shows how it is extended to $\mathbb{A}$.



In this thesis, the results of §§4 and 5 are proved in a general way which applies to any homogeneous structure $\mathbb{A}$ of atoms over finite relational vocabulary. In turn, the results of §7 related to register automata are proven for equality atoms $\mathbb{A} = \langle \mathbb{N}, = \rangle$. The criteria a structure $\mathbb{A}$ must meet to enable a generalisation of these results are detailed in §7.1.2.

Sets with atoms

For a fixed structure of atoms $\mathbb{A}$, we consider the universe of *sets with atoms* denoted by $V^{\mathbb{A}}$. Intuitively, these are sets whose elements are either atoms or 'simpler' sets of atoms. We define it formally in von-Neumann-like fashion, by means of a cumulative hierarchy arising through transfinite induction. Each level $V^{\mathbb{A}}_{\lambda}$ in that hierarchy is labelled with some ordinal number λ called its *rank*. The base level $V^{\mathbb{A}}_0 = \varnothing$ is just the empty set. For an ordinal number λ, the level of rank $\lambda + 1$ is

$$V^{\mathbb{A}}_{\lambda+1} = \mathcal{P}(V^{\mathbb{A}}_{\lambda}) \,\dot{\cup}\, \mathbb{A},$$

where $\dot{\cup}$ is the disjoint union. Finally, for a limit ordinal λ, we put

$$V^{\mathbb{A}}_{\lambda} = \bigcup_{\alpha<\lambda} V^{\mathbb{A}}_{\alpha}.$$

Example 2.4. initial levels of the hierarchy

Fix $\mathbb{A}$ to be equality atoms $\langle \mathbb{N}, = \rangle$, their elements being denoted as underlined numbers. We list a few unrolled definitions of initial levels of the hierarchy, as well as examples of sets belonging to them.

$$\begin{aligned}
V_0^{\mathbb{A}} &= \varnothing \\
V_1^{\mathbb{A}} &= \{\varnothing\} \,\dot\cup\, \mathbb{A}, & V_1^{\mathbb{A}} &\ni \varnothing, \underline{6} \\
V_2^{\mathbb{A}} &= \mathcal{P}(\{\varnothing\} \,\dot\cup\, \mathbb{A}) \,\dot\cup\, \mathbb{A} & V_2^{\mathbb{A}} &\ni \varnothing, \underline{6}, \{\varnothing, \underline{5}, \underline{9}\} \\
V_3^{\mathbb{A}} &= \mathcal{P}(\mathcal{P}(\{\varnothing\} \,\dot\cup\, \mathbb{A}) \,\dot\cup\, \mathbb{A}) \,\dot\cup\, \mathbb{A} & V_3^{\mathbb{A}} &\ni \varnothing, \underline{6}, \{\varnothing, \underline{5}, \underline{9}\}, \{\varnothing, \underline{7}, \{\varnothing, \underline{5}, \underline{9}\}\}
\end{aligned}$$

2.3.2 Atom automorphisms

In the theory of sets with atoms, the notion of finiteness gives way to a weaker concept of so-called *orbit-finiteness*. Intuitively, a set is orbit-finite if, and only if, it contains only finitely many different elements up to automorphism of $\mathbb{A}$. Therefore, the automorphisms of the infinite atom structure $\mathbb{A}$ play a crucial role when working with sets with atoms—they are a means to harness the infinite objects and making them amenable to further analysis.

An *atom automorphism* is a bijection $\pi : \mathbb{A} \hookrightarrow\!\!\!\!\rightarrow \mathbb{A}$ which preserves in both ways all relations of $\mathbb{A}$. By $\mathsf{Aut}(\mathbb{A})$ we denote the set of all automorphisms of the structure $\mathbb{A}$.

Let $S \subseteq \mathbb{A}$ be a (possibly empty) finite set of atoms. An *S-automorphism* is an atom automorphism $\pi \in \mathsf{Aut}(\mathbb{A})$ which additionally is the identity on S, i.e.

$$\pi(\alpha) = \alpha \quad \text{for every } \alpha \in S.$$

Let $\mathsf{Aut}_S(\mathbb{A})$ denote the set of all S-automorphisms of $\mathbb{A}$. Naturally, $\mathsf{Aut}(\mathbb{A}) = \mathsf{Aut}_\varnothing(\mathbb{A})$.

Example 2.5. automorphims of equality atoms

For $\mathbb{A}$ being equality atoms, the set $\mathsf{Aut}(\mathbb{A})$ is simply a set of all bijections $\mathbb{A} \hookrightarrow\!\!\!\!\rightarrow \mathbb{A}$, as there are no relations to preserve, except for the equality.

Example 2.6. automorphims of densely ordered atoms

For $\mathbb{A}$ being densely ordered atoms, automorphisms are monotonic bijections

$$\mathsf{Aut}(\mathbb{A}) = \{\pi : \mathbb{A} \hookrightarrow\!\!\!\!\rightarrow \mathbb{A} \mid \forall_{\alpha,\beta\in\mathbb{A}} \,.\, \alpha \leq \beta \Rightarrow \pi(\alpha) \leq \pi(\beta)\}.$$

Example 2.7. automorphims of timed atoms

For $\mathbb{A}$ being timed atoms, if we put $S = \{0\}$, any S-automorphism π of $\mathbb{A}$ must satisfy $\pi(0) = 0$ and $\pi(p+1) = \pi(p) + 1$ for every $p \in \mathbb{Q}$. Therefore, any monotone bijection on the unit interval $[0,1) \hookrightarrow\!\!\!\!\rightarrow [0,1)$ uniquely determines an S-automorphism, and each S-automorphism can be defined this way.

Atom automorphisms can be applied to sets with atoms. For $X \in V^{\mathbb{A}}$, $S \subseteq \mathbb{A}$ and $\pi \in \mathsf{Aut}_S(\mathbb{A})$, the set $\pi(X)$ is constructed from X by replacing every atom $\alpha \in \mathbb{A}$ by $\pi(\alpha) \in \mathbb{A}$. More formally, π acts on X point-wise as $\pi(X) = \{\pi(x) \mid x \in X\}$.

Later in the paper, there will be several use cases for this operation; we present here some of them. They all rely on standard set-theoretic encodings of mathematical objects (i.e., tuples, functions, etc.).

Example 2.8. applying atom automorphisms to data words

Let Σ be a finite alphabet. A finite *data word* is a sequence

$$w = (\sigma_0, \alpha_0) \cdots (\sigma_n, \alpha_n) \in (\Sigma \times \mathbb{A})^* \tag{2.1}$$

of pairs (σ_i, α_i) consisting of an input letter $\sigma_i \in \Sigma$ and an atom $\alpha_i \in \mathbb{A}$. An automorphism π acts on a data word w as above point-wise

$$\pi(w) = (\sigma_0, \pi(\alpha_0)) \cdots (\sigma_n, \pi(\alpha_n))$$

Example 2.9. applying atom automorphisms to register valuations

Let $\mathtt{X}$ be a finite set of register names and let $\mathbb{A}_\perp = \mathbb{A} \cup \{\perp\}$. where $\perp \notin \mathbb{A}$ represent an undefined value. A *register valuation* is a mapping $\mu : \mathtt{X} \to \mathbb{A}_\perp$, written $\mu : \mathbb{A}_\perp^{\mathtt{X}}$, assigning an atom $\mu(x)$ (or $\perp$) to every register $x \in \mathtt{X}$. An automorphism π acts on a register valuation μ as $\pi(\mu)(x) = \pi(\mu(x))$ for every $x \in \mathtt{X}$, i.e., $\pi(\mu) = \pi \circ \mu$, where we assume that $\pi(\perp) = \perp$.

2.3.3 Invariance

A set with atoms X is *S-invariant* if $\pi(X) = X$ for every S-automorphism $\pi \in \mathsf{Aut}_S(\mathbb{A})$. A slight reformulation of this definition leads us to an alternative statement:

Fact 2.10.

Set X is S-invariant if, and only if the for every $\pi \in \mathsf{Aut}_S(\mathbb{A})$:

$$x \in X \Leftrightarrow \pi(x) \in X.$$

Example 2.11.

Note that π does not need to be the identity on X for X to be S-invariant. The set $\mathbb{N}$ is $\varnothing$-invariant. Let $\pi : \mathbb{N} \hookrightarrow\!\!\!\rightarrow \mathbb{N}$ be a bijection between the even and odd numbers such that for every $n \in \mathbb{N}$:

$$\begin{aligned} 2n &\overset{\pi}{\mapsto} 2n+1 \\ 2n+1 &\overset{\pi}{\mapsto} 2n \end{aligned}$$

Observe that $\pi(\mathbb{N}) = \mathbb{N}$ even though π has no fixpoints.

A set X is *invariant*[4] if it is S-invariant with $S = \varnothing$.

Orbits

The *S-orbit* of an element $x \in X$ is the set of all elements $\pi(x)$ which can be obtained by applying some S-automorphism π to x:

$$\mathsf{orbit}_S(x) := \{\pi(x) \in X \mid \pi \in \mathsf{Aut}_S(\mathbb{A})\}$$

Note that x can be an arbitrary object on which the action of automorphisms is defined. The *orbit* of x is just its S-orbit with $S = \varnothing$, written $\mathsf{orbit}(x)$.

The set of all S-orbits of elements of X is denoted as $\mathsf{Orbits}_S(X) := \{\mathsf{orbit}_S(x) \mid x \in X\}$.

Fact 2.12.

Elements $x, y \in X$ have the same S-orbit $\mathsf{orbit}_S(x) = \mathsf{orbit}_S(y)$ if, and only if, $\pi(x) = y$ for some $\pi \in \mathsf{Aut}_S(\mathbb{A})$.

The *S-closure* of a set X is the union of the S-orbits of its elements:

$$\mathsf{closure}_S(X) := \bigcup_{x \in X} \mathsf{orbit}_S(x)$$

In particular, the S-orbit of x is the S-closure of the singleton set $\{x\}$: $\mathsf{orbit}_S(x) = \mathsf{closure}_S(\{x\})$. For brevity, we call $\varnothing$-closure $\mathsf{closure}(X)$ simply a *closure*. The following fact characterises invariance in terms of closures.

Fact 2.13.

A set X is S-invariant if, and only if, $\mathsf{closure}_S(X) = X$.

Proof of fact 2.13.

The 'only if' direction follows from the definition of S-invariance. For the 'if' direction, observe that $\mathsf{closure}_S(X) = X$ implies $\pi(X) \subseteq X$ for every S-automorphism π. The opposite inclusion stems from S-automorphisms' closure under inverse: $\pi^{-1}(X) \subseteq X$, hence $X \subseteq \pi(X)$. □

2.4 Register automata (RA)

Equipped with the basics of the theory of sets with atoms, we are ready to move to *register automata* (RA). The model is parametrised by the choice of atoms—let us fix an arbitrary relational structure $\mathbb{A}$ over a finite signature σ for the purposes of the rest of §2.4.

A *data word* over the finite alphabet Σ is a sequence of letters from $\Sigma \times \mathbb{A}$. Sets of all finite and infinite data words are denoted as $(\Sigma \times \mathbb{A})^*$ and $(\Sigma \times \mathbb{A})^\omega$, respectively.

[4] A term *equivariant* is also often used in the literature instead of *invariant*. Also, in the case of S-invariant X, the literature often calls S a *support* of X, see e.g. [10, 11].

A language $L \subseteq (\Sigma \times \mathbb{A})^*$ of data words is an arbitrary set of finite data words, while ω-language is am arbitrary set of infinite data words.

Register automaton $\mathcal{A}$, will have its semantics $[\![\mathcal{A}]\!]$ defined in terms of LTS. Therefore, it can be seen as an acceptor of languages of both finite and infinite data words, depending on the chosen notion of the language $\mathcal{L}([\![\mathcal{A}]\!])$, $\mathcal{L}^\omega([\![\mathcal{A}]\!])$ (cf. §2.2).

We begin by introducing a notion of *register constraints*. It is followed by formal definitions of three different variants of the model, namely:

- nondeterministic register automata with guessing ($\mathsf{NRA^g}$); the most expressive variant,
- nondeterministic register automata without guessing (NRA),
- deterministic register automata (DRA); the least expressive model.

The weaker classes will arise as syntactic limitations of the stronger ones. Lastly, we will provide some definitions common to these classes and consider some modifications/extensions of RA.

2.4.1 Register constraints

A *register constraint* is a quantifier-free formula φ generated by the grammar

$$\varphi, \psi \ ::\equiv \mathsf{TRUE} \mid \mathsf{FALSE} \mid \mathtt{x}_1 = \mathtt{x}_2 \mid \mathtt{x}_1 = \bot \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \tag{2.2}$$

$$\langle \mathtt{x}_1, \mathtt{x}_2, \ldots, \mathtt{x}_{\overline{R}} \rangle \in R \quad \text{for every relational symbol } R \text{ in } \sigma \tag{2.3}$$

where $\mathtt{x}_\bullet$ are variables, $\bot$ is a special constant denoting an undefined data value, and $\overline{R}$ denotes the arity of R. In the case of equality atoms $\mathbb{A} = \langle \mathbb{N}, = \rangle$, there are no relations other than equality, so the rule (2.3) is not used. We denote the set of constraints with variables from $\mathtt{X}$ as $\textsc{Constr}(\mathtt{X})$.

The *semantics* of a constraint $\varphi(x_1, \ldots, x_n)$ with n free variables $\mathtt{X} = \{\mathtt{x}_1, \ldots, \mathtt{x}_n\}$ is the set defined by it, consisting of valuations satisfying it: $[\![\varphi]\!] = \{\mu \in \mathbb{A}_\bot^{\mathtt{X}} \mid \mu \models \varphi\}$. Using [10, Lemma 7.5] we easily deduce:

Claim 2.14. constraints represent invariant subsets

Let $\mathbb{A}$ be a homogeneous structure of atoms. Subsets of $\mathbb{A}_\bot^{\mathtt{X}}$ definable by constraints are exactly invariant subsets of $\mathbb{A}_\bot^{\mathtt{X}}$.

Maximal register constraints

Conceptually, a *maximal register constraint* is a restricted form of a constraint, which is maximally specific, non-redundant, and which defines some minimal non-empty invariant subset of $\mathbb{A}_\bot^{\mathtt{X}}$—i.e., an $\mathsf{orbit}(\mu)$ of some $\mu \in \mathbb{A}_\bot^{\mathtt{X}}$. In the next sections, such constraints will find use as a finite representation of those orbits.

Fix $\mu \in \mathbb{A}_\bot^{\mathtt{X}}$. We define the maximal constraint $\hat{\varphi}_\mu$ corresponding to $\mathsf{orbit}(\mu)$ as the conjunction of atomic formulas

$$\bigwedge_{R \in \sigma} \ \bigwedge_{\mathtt{x}_1, \ldots, \mathtt{x}_{\overline{R}} \in \mathtt{X}} \begin{cases} \langle \mathtt{x}_1, \ldots, \mathtt{x}_{\overline{R}} \rangle \in R & \text{if } (\mu(\mathtt{x}_1), \ldots, \mu(\mathtt{x}_{\overline{R}})) \in R \\ \neg\langle \mathtt{x}_1, \ldots, \mathtt{x}_{\overline{R}} \rangle \in R & \text{otherwise} \end{cases} .$$

Such a constraint exhaustively specifies all the relations that hold between the atoms present in a valuation μ. We denote the set of all maximal constraints with variables from $\mathtt{X}$ as $\textsc{MaxConstr}(\mathtt{X})$.

Claim 2.15. orbits of valuations correspond to maximal constraints

Let $\mathbb{A}$ be a homogeneous structure of atoms. For every $\mu \in \mathbb{A}_{\perp}^{\mathtt{X}}$ we have that $[\![\hat{\varphi}_\mu]\!] = \mathsf{orbit}(\mu)$

Proof of claim 2.15.

For the right-to-left inclusion observe that $\mu \in [\![\hat{\varphi}_\mu]\!]$, and $[\![\hat{\varphi}_\mu]\!]$ is invariant by claim 2.14, so $\mathsf{orbit}(\mu) \subseteq [\![\hat{\varphi}_\mu]\!]$.

Let us assume $[\![\hat{\varphi}_\mu]\!] \setminus \mathsf{orbit}(\mu)$ is not empty and contains some μ'. Since $\mu' \notin \mathsf{orbit}(\mu)$, there exists no automorphism π such that $\mu = \pi(\mu')$. This mismatch between μ and μ' is evidenced by a relation R and a tuple of variables $v = (\mathtt{x}_1, \ldots, \mathtt{x}_{\overline{R}}) \in \mathtt{X}$ such that $(\mu(\mathtt{x}_1), \ldots, \mu(\mathtt{x}_{\overline{R}}))$ is in R but $(\mu'(\mathtt{x}_1), \ldots, \mu'(\mathtt{x}_{\overline{R}}))$ is not, or vice versa. Therefore, $\mu' \notin [\![\hat{\varphi}_\mu]\!]$, a contradiction. □

Corollary 2.16.

For homogeneous atoms, there is a bijective correspondence between constraints $\hat{\varphi}_\mu$ and orbits of valuations $\mathsf{orbit}(\mu)$.

We denote the orbit corresponding to some maximal constraint φ as $\mathsf{orbit}(\varphi)$.

2.4.2 Nondeterministic register automata with guessing ($\textsf{NRA}^{\mathsf{g}}$)

Let Σ be a finite alphabet. A *nondeterministic register automaton with guessing* ($\textsf{NRA}^{\mathsf{g}}$) with $k \in \mathbb{N}$ registers is a tuple $\mathcal{A} = (\mathtt{X}, \Sigma, L, L_{\mathsf{ini}}, L_{\mathsf{fin}}, \Delta)$ where:

- Σ is a finite alphabet,
- $\mathtt{X} = \{\mathtt{x}_1, \mathtt{x}_2, \ldots, \mathtt{x}_k\}$ is a finite set of register names,
- L is a finite set of *control locations*, of which $L_{\mathsf{ini}}, L_{\mathsf{fin}} \subseteq L$ are marked as *initial* and *final*, respectively,
- Δ is a set of *transition rules*.

A rule of the form

$$(p, \sigma, \varphi, q) \in \Delta \quad \subseteq \quad L \times \Sigma \times \textsc{Constr}(\mathtt{Z}) \times L \tag{2.4}$$

also written as

$$(p \xRightarrow{\sigma, \varphi} q) \qquad\qquad \text{(if } \varphi = \texttt{TRUE}\text{, it can be omitted)}$$

indicates that the automaton can go from control location $p \in L$ to $q \in L$ by reading input symbol $\sigma \in \Sigma$, provided that the transition constraint φ is satisfied in the current configuration.

Above, the set $\mathtt{Z}$ of variables that can appear in constraint φ is $\mathtt{X} \cup \{\mathtt{x}'_1, \mathtt{x}'_2, \ldots, \mathtt{x}'_k\} \cup \{\mathtt{y}\}$. Intuitively, $\varphi(\mathtt{x}_1, \ldots, \mathtt{x}_k,\ \mathtt{x}'_1, \ldots, \mathtt{x}'_k, \mathtt{y})$ relates the registers' values before ($\mathtt{x}_\bullet$) and after a transition ($\mathtt{x}'_\bullet$) with the currently read atom $\mathtt{y}$.

Intuition

An NRA$^{\mathsf{g}}$ reads an input data word $w \in (\Sigma \times \mathbb{A})^*$ letter by letter. To handle the atoms found at the input, the automaton is given a set of registers, in which it can store them. Transition execution is subject to satisfying its constraint. Such condition speaks about the newly read atom and atoms stored in registers (before and after the transition). The constraint may use any symbol from the atoms' signature σ.

Semantics

As it was already announced in §2.2, we will formally define the semantics of a register automaton $\mathcal{A}$ as above in terms of an infinite labeled transition system $[\![\mathcal{A}]\!] = (C, \Gamma, C_{\mathsf{ini}}, C_{\mathsf{fin}}, \rightarrow)$. Since there is no danger of confusion, we use $[\![\mathcal{A}]\!]$ to denote both the labeled transition system above and its domain. Recall the previous definition of the set of configurations and labels:

LTS for a k-register automaton

labels	$\Gamma = \Sigma \times \mathbb{A}$
configurations	$C = L \times \mathbb{A}_\perp^{\mathtt{X}}$ pairs consisting of a control location and a register valuation

Above, we used the notation $\mathbb{A}_\perp = \mathbb{A} \cup \{\perp\}$ introduced previously in the example 2.9. We complete the construction as follows:

initial configurations	$C_{\mathsf{ini}} = L_{\mathsf{ini}} \times \{\lambda x.\perp\} \subseteq C$ pairs (p, μ) with $p \in L_I$ and $\mu(x) = \perp$ for all registers $x \in \mathtt{X}$,
final configurations	$C_{\mathsf{fin}} = L_{\mathsf{fin}} \times \mathbb{A}_\perp^{\mathtt{X}}$ pairs (p, μ) with $p \in L_F$; without any further restriction on μ

Lastly, there is a transition $(p, \mu) \xrightarrow{\sigma,\alpha} (q, \mu')$ between two configurations if, and only if there exists a transition rule $(p, \sigma, \varphi, q) \in \Delta$ such that $\eta \models \varphi$, where $\eta : \mathtt{Z} \to \mathbb{A}_\perp$ is a valuation defined as follows:

$$\mathtt{x}_i \overset{\eta}{\mapsto} \mu(\mathtt{x}_i) \text{ and } \mathtt{x}'_i \overset{\eta}{\mapsto} \mu'(\mathtt{x}_i) \text{ for each } i \in \{1, 2, \ldots, k\}$$
$$\mathtt{y} \overset{\eta}{\mapsto} \alpha$$

In other words, the constraint φ must be true when substituting the register values from μ, μ' for the variables $\mathtt{x}_\bullet, \mathtt{x}'_\bullet$, and the current atom α for the variable $\mathtt{y}$.

We use separate notations for transition rules of $\mathcal{A}$ and the induced transitions of $[\![\mathcal{A}]\!]$:

$$(p \overset{\sigma,\varphi}{\Longrightarrow} q) \in \Delta \qquad\qquad (p, \mu) \xrightarrow{\sigma,\alpha} (q, \mu') \in \rightarrow.$$

Each run π of $\mathcal{A}$

$$\pi = (l_0, \mu_0) \xrightarrow{\sigma_1,\alpha_1} (l_1, \mu_1) \xrightarrow{\sigma_2,\alpha_2} (l_2, \mu_2) \xrightarrow{\sigma_3,\alpha_3} \dots$$

has at least one *designating sequence of transition rules* $\rho \in \Delta^*$ of the form

$$\rho = l_0 \xRightarrow{\sigma_1,\varphi_1} l_1 \xRightarrow{\sigma_2,\varphi_2} l_2 \xRightarrow{\sigma_3,\varphi_3} \dots$$

with the property that the kth transition $(l_{k-1}, \mu_{k-1}) \xrightarrow{\sigma_k,\alpha_k} (l_k, \mu_k)$ of π is induced by the kth rule $l_{k-1} \xRightarrow{\sigma_k,\varphi_k} l_k$ of σ for all k.

We define the language $\mathcal{L}(\mathcal{A})$ of the automaton $\mathcal{A}$ to be the language of its LTS $[\![\mathcal{A}]\!]$. Therefore, all definitions from §2.2 can be naturally instantiated in the setting of NRA, including the notions of run, an accepting run, language of a configuration, and reachable configurations. Words over the alphabet $\Sigma \times \mathbb{A}$ will be called *data words*, as in example 2.9.

2.4.3 Nondeterministic register automata without guessing (NRA)

Roughly speaking, an automaton in $\mathsf{NRA}^{\mathsf{g}}$ (i.e., with guessing) can nondeterministically 'guess' an atom which is not stored in any of its registers and was not read from the input. For example, the constraint φ may require the new register value $\mathtt{x}'_1$ to be different both from the old register values $\mathtt{x}_1, \mathtt{x}_2, \dots, \mathtt{x}_k$ and from the atom in the input $\mathtt{y}$. The class of *nondeterministic register automata without guessing* (NRA) arises as a simple syntactical restriction of $\mathsf{NRA}^{\mathsf{g}}$, which eliminates this type of behaviour. More precisely, in NRA every constraint imposes an additional fixed requirement for the atoms stored in register:

$$\bigwedge_{i\in\{1,\dots,k\}} \mathtt{x}'_i = \mathtt{x}_i \vee \mathtt{x}'_i = \mathtt{y}.$$

In words, any new value of a register can only come from the input; otherwise, an old value of the register is preserved.

The shape of transition rules inherited from $\mathsf{NRA}^{\mathsf{g}}$, in the presence of such a limiting restriction, may be seen a bit superfluous and unnecessarily convoluted to work with. Therefore, we simplify it to

$$(p, \sigma, \varphi, \mathtt{Y}, q) \quad \in \quad L \times \Sigma \times \textsf{Constr}(\mathtt{X} \cup \{\mathtt{y}\}) \times \mathcal{P}(\mathtt{X}) \times L$$

written also as

$$(p \xRightarrow{\sigma,\varphi,\mathtt{Y}} q) \qquad \text{(if } \varphi = \mathsf{TRUE} \text{ or } \mathtt{Y} = \varnothing\text{, they can be omitted, respectively).}$$

Here, the constraint only features $k+1$ variables corresponding to current values of the registers and the atom at the input. The *reset set* $\mathtt{Y} \subseteq \mathtt{X}$ is a set specifying registers to which the newly read atom is to be assigned, the rest retaining their value. More formally, the above transition is interpreted as (p, σ, φ', q) where φ' has the form:

$$\varphi \wedge \bigwedge_{i\in\{1,\dots,k\}} \begin{cases} \mathtt{x}'_i = \mathtt{y} & \text{if } \mathtt{x}_i \in \mathtt{Y} \\ \mathtt{x}'_i = \mathtt{x}_i & \text{otherwise} \end{cases}$$

2.4.4 Deterministic register automata (DRA)

In order to define the last class, we translate the notion of *deterministic* LTS introduced in §2.2 to the concrete setting of NRA. A register automaton without guessing $\mathcal{A}$ is *deterministic* if it has precisely one initial location $L_{\mathsf{ini}} = \{p_{\mathsf{ini}}\}$ and, for every two rules $(p, \sigma, \varphi, \mathtt{Y}, q)$ and $(p, \sigma, \varphi', \mathtt{Y}', q')$ starting in the same location p, over the same input symbol σ and with overlapping guards $[\![\varphi \wedge \varphi']\!] \neq \varnothing$, we necessarily have $\mathtt{Y} = \mathtt{Y}'$ and $q = q'$. Hence $\mathcal{A}$ has at most one run over every data word w. We define *deterministic register automata* (DRA) as the subset of NRA which are deterministic.

A DRA can be easily transformed into a *total* one, i.e., one where for every location $p \in L$ and input symbol $\sigma \in \Sigma$, the sets defined by the constraints $\{[\![\varphi]\!] \mid \exists \mathtt{Y}, q \,.\, p \xrightarrow{\sigma,\varphi,\mathtt{Y}} q\}$ are a partition of all register valuations $\mathbb{A}_\perp^{\mathtt{X}}$. Thus, a total DRA has exactly one run over every timed word w.

Example 2.17.

Let $\Sigma = \{\sigma\}$ be a unary alphabet. As an example of a language L recognised by an NRA_1, but not by any DRA, consider the set of data words where the last atom reappears earlier, i.e., words of the form: $(\sigma, \alpha_1) \cdots (\sigma, \alpha_n)$ where $\alpha_i = \alpha_n$ for some $1 \leq i < n$. The language L is recognised by the NRA_1 $\mathcal{A} = (\mathtt{X}, \Sigma, L, L_{\mathsf{ini}}, L_{\mathsf{fin}}, \Delta)$ with one register $\mathtt{X} = \{\mathtt{x}\}$ and three locations $L = \{p, q, r\}$, of which $L_{\mathsf{ini}} = \{p\}$ is initial and $L_{\mathsf{fin}} = \{r\}$ is final, and transition rules

$$(p \xRightarrow{\sigma} p) \qquad (p \xRightarrow{\sigma,\{\mathtt{x}\}} q) \qquad (q \xRightarrow{\sigma,\mathtt{x}\neq\mathtt{y}} q) \qquad (q \xRightarrow{\sigma,\mathtt{x}=\mathtt{y}} r).$$

Intuitively, the automaton waits in p until it guesses that the subsequent input will be the last appearance of the atom α_i (not counting α_n), at which point it moves to q by storing α_i in the register. From q, the automaton can accept by going to r exactly when the atom stored in the register reappears in the input. The language L is not recognised by any DRA since, intuitively, any deterministic acceptor needs to store unboundedly many different atoms α_i.

Notations

We use the following notations:

- NRA_k^g, NRA_k, and DRA_k denote the classes of k-register $\mathsf{NRA}^{\mathsf{g}}$, NRA, and DRA, respectively,
- we say that a data language is an $\mathsf{NRA}^{\mathsf{g}}$ language, DRA language, NRA_k language, etc., if it is recognised by a register automaton of the respective type,
- additionally, when it is clear from the context, for a class C and a language L we write $L \in C$, to denote that L is recognised by DRA,
- classes of register automata over alphabet Σ are denoted as $\mathsf{NRA}(\Sigma)$, $\mathsf{NRA}_k(\Sigma)$, $\mathsf{DRA}_k(\Sigma)$.

2.4.5 One-register automata

Nondeterministic register automata with just one register enjoy stronger algorithmic properties than the full class of nondeterministic register automata. It was already observed in Kaminski and Francez's seminal paper that the inclusion problem becomes decidable[5].

Theorem 2.18. c.f. [61, Appendix A]

For $A \in \mathsf{NRA}$ and $B \in \mathsf{NRA}_1$ the language inclusion problem $\mathcal{L}(A) \subseteq \mathcal{L}(B)$ is decidable.

We immediately obtain the following corollary, which we will use in §7.1.

Corollary 2.19.

For $A \in \mathsf{DRA}$ and $B \in \mathsf{NRA}_1$ the language equality problem $\mathcal{L}(A) = \mathcal{L}(B)$ is decidable.

Proof of corollary 2.19.

The inclusion $\mathcal{L}(A) \subseteq \mathcal{L}(B)$ can be checked as a special instance of theorem 2.18. The reverse inclusion $\mathcal{L}(B) \subseteq \mathcal{L}(A)$ reduces to checking emptiness of a product construction of B with the complement of A. $\square$

2.4.6 Invariance of register automata

The following lemma expresses the fundamental invariance properties of register automata. Given a valuation μ of registers $\mathtt{X}$, by $\mu(\mathtt{X}) \subseteq \mathbb{A}$ we mean the set of atoms stored in registers: $\mu(\mathtt{X}) = \{\mu(\mathtt{x}) \mid \mathtt{x} \in \mathtt{X},\ \mu(\mathtt{x}) \in \mathbb{A}\}$. Automorphisms act on configurations by preserving the control location: $\pi(p, \mu) = (p, \pi(\mu)) = (p, \pi \circ \mu)$.

Lemma 2.20. Invariance of NRA

1. The transition system $[\![A]\!]$ is invariant: If $c \xrightarrow{\sigma,\alpha} d$ in $[\![A]\!]$ and π is an automorphism, then $\pi(c) \xrightarrow{\sigma,\pi(\alpha)} \pi(d)$ in $[\![A]\!]$.
2. The function $\mathcal{L}(\_)$ mapping a configuration c to the language $\mathcal{L}(c)$ it recognises from c is invariant: For all automorphisms π, $\mathcal{L}(\pi(c)) = \pi(\mathcal{L}(c))$.
3. The language $\mathcal{L}(p, \mu)$ recognised from a configuration (p, μ) is $\mu(\mathtt{X})$-invariant: For all $\mu(\mathtt{X})$-automorphims π, $\pi(\mathcal{L}(p, \mu)) = \mathcal{L}(p, \mu)$.

We refrain from proving lemma 2.20, since proofs of analogous invariance properties, in the more involved setting of timed automata, are provided later (facts 7.19 to 7.21 in §7.2.1). For the proof of (1) in the setting of equality atoms, we refer the reader to [10, Sect. 1.1]; the other points are readily derivable from (1).

[5] A *window* in the terminology of [61] corresponds to a register in this paper's terminology. [61, Appendix A] shows that the inclusion problem $\mathcal{L}(A) \subseteq \mathcal{L}(B)$ is decidable when B is a two-window automaton. Due to the semantics of window reassignment of [61], two-window automata are of intermediate expressive power between one-register automata and two-register automata.

2.4.7 Varieties of register automata

Register automata with output

Deterministic and total register automata, enriched with output labels on transition rules coning from a finite alphabet Γ, are called register automata with output, their class being denoted as $\mathsf{DRA}(\Sigma, \Gamma)$. Formally, $\mathcal{A} \in \mathsf{DRA}(\Sigma, \Gamma)$ is a tuple $(\mathtt{X}, \Sigma, \Gamma, L, L_{\mathsf{ini}}, \delta)$ with $\delta \subseteq L \times \Sigma \times \Gamma \times L$, such that:

- $(\mathtt{X}, \Sigma, L, L_{\mathsf{ini}}, \varnothing, \delta') \in \mathsf{DRA}(\Sigma)$, where δ' contains projections of tuples of δ ignoring the output letter Γ component,
- input label together with initial and final states of a transition rule in δ uniquely determine an output letter $\gamma \in \Gamma$.

Semantics of register automata with output have semantics defined by analogy to regular register automata, the additional output letter appearing on all transitions originating from a given transition rule. For automaton $\mathcal{A} \in \mathsf{DRA}(\Sigma, \Gamma)$, $[\![\mathcal{A}]\!] \in \mathsf{DLTS}(\Sigma, \Gamma)$. For every word $w \in (\Sigma \times \mathbb{A})^* \cup (\Sigma \times \mathbb{A})^\omega$ (both finite and infinite), $\mathcal{A}$ outputs an unique word $\mathcal{A}(w) := [\![\mathcal{A}]\!](w)$.

Automata with maximal constraints

To each of the classes of automata mentioned above—$\mathsf{NRA}^{\mathsf{g}}$, NRA and DRA—we may add a simple structural requirement for transition rules to be labelled exclusively using maximal constraints. This gives rise to corresponding subclasses *with maximal constraints*, denoted by additional superscript $\bullet^{\mathsf{max}}$. This special form of automata— will find usage in the sections of this dissertation devoted to synthesis problem. In particular, we will need the notion of the automaton with output using maximal constraints, denoted as $\mathsf{DRA}^{\mathsf{max}}(\Sigma, \Gamma)$.

Lemma 2.21.

For every $C \in \{\mathsf{NRA}^{\mathsf{g}}, \mathsf{NRA}, \mathsf{DRA}\}$ and $\mathcal{A} \in C$ there exists an automaton $\mathcal{A}' \in C^{\mathsf{max}}$ with maximal constraints having the same number of registers and such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.

Proof of lemma 2.21.

Fix an automaton $\mathcal{A} = (\mathtt{X}, \Sigma, L, L_{\mathsf{ini}}, L_{\mathsf{fin}}, \delta) \in \mathsf{NRA}^{\mathsf{g}}$. To obtain $\mathcal{A}'$, replace every transition rule labelled with a non-maximal constraint φ with an equivalent set of its copies using maximal constraints. More precisely, for every $(p, \sigma, \varphi, q) \in \delta$ relation δ' contains its copies $(p, \sigma, \varphi_i, q)$, where orbits corresponding to $\varphi_i \in \textsc{MaxConstr}(\mathtt{X} \dot\cup \mathtt{X} \cup \{\mathtt{y}\})$ form a partition of $[\![\varphi]\!]$.

It is easy to see that $\mathcal{A}'$ accepts exactly the same words as $\mathcal{A}$. Moreover, if $\mathcal{A}$ is without guessing or deterministic, $\mathcal{A}$ is so, too. $\square$

Epsilon transitions

One of the extensions of nondeterministic register automata, the addition of *ε-transition rules*, often proves useful for simplifying the construction of an automaton.

Intuitively, an ε-transition rule—one labeled with a special symbol ε—can be executed without reading any symbol of the input word. While each ordinary transition is corresponds to a single letter of the input word, the automaton can make arbitrary number of ε-transitions, either before reading the first letter, between letters of the word being read, or after it has ended (in case of finite words).

It turns out that $\mathsf{NRA}^{\mathrm{g}}$ and NRA with ε-transition rules are as expressive as respective models without ε-transition rules. One can remove them by using a straightforward construction:

Lemma 2.22. [10, excercise 2.]

For every $\mathsf{NRA}^{\mathrm{g}}$ with ε-transition rules one can effectively build an $\mathsf{NRA}^{\mathrm{g}}$ without them and the same number of registers recognising the same language. The same holds also for NRA.

For this reason, in this dissertation, we deal exclusively with NRA without ε-transition rules. However, the reader should keep in mind that all the results concerning register automata easily extend to them.

2.5 Timed automata (TA)

Timed automata are very similar to register automata instantiated with timed atoms $\mathbb{A} = \langle \mathbb{Q}, \leq, +1 \rangle$ [12]. Both models share many properties: e.g., a strong link concerning a class of language-related decision problems was provided in [45]. However, there are several minor but not negligible differences between the models that force us to treat timed automata individually:

- timed atoms lack homogeneity[6] (this can be seen as the principal source of differences),
- certain syntactic and semantic restrictions are traditionally imposed on timed languages and timed automata, including monotonicity of time, nonnegative timestamps, the special status of the initial timestamp 0, and the concrete syntax of transition constraints.

For the reasons above, timed automata can be seen only as a strict subclass of the corresponding register model. In this dissertation, we avoid defining timed automata using a generic register model, and instead introduce them separately.

[6] For instance, the set of pairs of timed atoms consists of infinitely many orbits.

2.5.1 Timed words and languages

Fix a finite alphabet Σ. Timed words are obtained by instantiating data words to timed atoms $\mathbb{A} = \langle \mathbb{Q}, \leq, +1 \rangle$ [7], and imposing additional conditions—non-negativeness and monotonicity.

A *timed word* over Σ is any sequence of the form

$$w \;=\; (\sigma_1, \tau_1) \ldots (\sigma_n, \tau_n) \;\in\; (\Sigma \times \mathbb{Q}_{\geq 0})^* \tag{2.5}$$

which is *monotonic*, in the sense that the timed atoms (called *timestamps* henceforth) $\tau_\bullet$ satisfy $0 \leq \tau_1 \leq \tau_2 \leq \cdots \leq \tau_n$. For a timed word w as in eq. (2.5) and an increment $\delta \in \mathbb{Q}_{\geq 0}$, let $w + \delta = (\sigma_1, \tau_1 + \delta) \ldots (\sigma_n, \tau_n + \delta)$ be the timed word obtained from w by increasing all timestamps by δ. Let $\mathbb{T}(\Sigma)$ be the set of all timed words over Σ, and let $\mathbb{T}_{\geq \tau}(\Sigma)$ be, for $\tau \in \mathbb{Q}_{\geq 0}$, the set of timed words with $\tau_1 \geq \tau$. A *timed language* is any subset of $\mathbb{T}(\Sigma)$.

The concatenation $w \cdot v$ of two timed words w and v is defined only when the first timestamp of v is greater or equal than the last timestamp of w. Using this partial operation, we define, for a timed word $w \in \mathbb{T}(\Sigma)$ and a timed language $L \subseteq \mathbb{T}(\Sigma)$, the left quotient $w^{-1}L := \{v \in \mathbb{T}(\Sigma) \mid w \cdot v \in L\}$. Clearly $w^{-1}L \subseteq \mathbb{T}_{\geq \tau_n}(\Sigma)$.

2.5.2 Clock constraints and regions

Let $\mathtt{X} = \{\mathtt{x}_1, \ldots, \mathtt{x}_k\}$ be a finite set of clocks. A *clock valuation* is a function $\mu \in \mathbb{Q}_{\geq 0}^{\mathtt{X}}$ assigning a non-negative rational number $\mu(\mathtt{x})$ to every clock $\mathtt{x} \in \mathtt{X}$. A *clock constraint* is a quantifier-free formula of the form

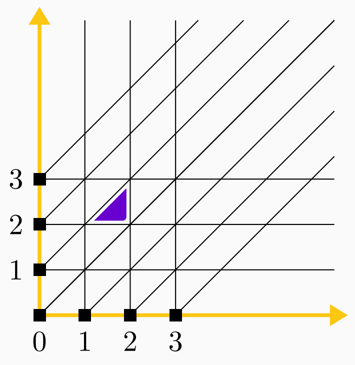
$$\varphi, \psi \;\;::\equiv\;\; \mathsf{TRUE} \mid \mathsf{FALSE} \mid \mathtt{x}_i - \mathtt{x}_j \sim z \mid \mathtt{x}_i \sim z \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi,$$

where '$\sim$' is a comparison operator in $\{<, \leq, >, \geq\}$ and $z \in \mathbb{Z}$. A clock valuation μ satisfies a constraint φ, written $\mu \models \varphi$, if interpreting each clock $\mathtt{x}_i$ by $\mu(\mathtt{x}_i)$ makes φ true. Let $[\![\varphi]\!]$ be the set of clock valuations $\mu \in \mathbb{Q}_{\geq 0}^{\mathtt{X}}$ such that $\mu \models \varphi$. If $[\![\varphi]\!]$ is non-empty and it does not strictly include a non-empty $[\![\psi]\!] \subsetneq [\![\varphi]\!]$ for some constraint ψ, then we say that it is a *region*. For example, for $\mathtt{X} = \{\mathtt{x}, \mathtt{y}\}$ we have that $r_0 = [\![1 < \mathtt{x} < 2 \wedge \mathtt{y} = 3]\!]$ is a region, while $r_1 = [\![1 < \mathtt{x} \leq 2 \wedge \mathtt{y} = 3]\!]$ is not. Nonetheless, the latter partitions into two regions $r_1 = r_0 \cup [\![\mathtt{x} = 2 \wedge \mathtt{y} = 3]\!]$, and we will later see that this is a general phenomenon. A *k, m-region* is a region $[\![\varphi]\!]$ where φ has k clocks and absolute value of maximal constant bounded by m. Let $\mathsf{Reg}(k, m)$ be a set of all k, m-regions.

Example 2.23. visualisation of regions for $k = 2$

For instance, the clock constraint $1 < \mathtt{x}_1 < 2 \;\wedge\; 2 < \mathtt{x}_2 < 3 \;\wedge\; \mathtt{x}_2 - \mathtt{x}_1 < 1$ defines a $2, 5$-region consisting of an open triangle with nodes $(1, 2)$, $(2, 2)$ and $(2, 3)$.

[7] Equivalently, the reals $\mathbb{R}$ could be considered in place of rationals.



A region $[\![\varphi]\!]$ is *bounded* if it is bounded as a subset of $\mathbb{Q}_{\geq 0}^{\mathtt{X}}$ in the classical sense, i.e., there exists $M \in \mathbb{Q}_{\geq 0}$ such that $[\![\varphi]\!] \subseteq [0, M]^{\mathtt{X}}$.

2.5.3 Nondeterministic timed automata (NTA)

A *nondeterministic timed automaton* (NTA) is a tuple $\mathcal{A} = (\mathtt{X}, \Sigma, L, L_{\mathsf{ini}}, L_{\mathsf{fin}}, \Delta)$, where:

- $\mathtt{X} = \{\mathtt{x}_1, \mathtt{x}_2, \ldots\}$ is a finite set of clocks,
- Σ is a finite input alphabet,
- L is a finite set of control locations,
- $L_{\mathsf{ini}}, L_{\mathsf{fin}} \subseteq L$ are the subsets of initial, resp., final, control locations, and
- Δ is a finite set of transition rules

Transition rules in turn have the form

$$(p, \sigma, \varphi, \mathtt{Y}, q) \qquad \text{written also as } (p \xrightarrow{\sigma, \varphi, \mathtt{Y}} q) \tag{2.6}$$

with $p, q \in L$ control locations, $\sigma \in \Sigma$, φ a clock constraint to be tested, and $\mathtt{Y} \subseteq \mathtt{X}$ the set of clocks to be reset. If $\mathtt{Y}$ is just the singleton $\{\mathtt{x}\}$, we sometimes write $\mathtt{x}$ instead of $\{\mathtt{x}\}$. Moreover, $\mathtt{Y}$ and φ can be omitted, if they are $\varnothing$ or $\bot$, respectively. We write NTA for the class of all nondeterministic timed automata, NTA_k when the number k of clocks is fixed, $\mathsf{NTA}_{\bullet,m}$ when the bound m on constants is fixed, and $\mathsf{NTA}_{k,m}$ when both k and m are fixed.

An $\mathsf{NTA}_{\bullet,m}$ $\mathcal{A}$ is *always resetting* if every transition rule resets some clock ($\mathtt{Y} \neq \varnothing$ in (2.6)), and *greedily resetting* if, for every clock $\mathtt{x}$, whenever φ implies that $\mathtt{x}$ belongs to $\{0, \ldots, m\} \cup (m, \infty)$, then $\mathtt{x} \in \mathtt{Y}$. Intuitively, a greedily resetting automaton resets every clock whose value is either an integer, or exceeds the maximal constant m.

Reset-point semantics

We introduce a semantics based on reset points instead of clock valuations. A *reset-point assignment* is a function $\mu \in \mathbb{Q}_{\geq 0}^{\mathtt{X}}$ storing, for each clock $\mathtt{x} \in \mathtt{X}$, the timestamp $\mu(\mathtt{x})$ when $\mathtt{x}$ was last reset. Reset-point assignments and clock valuations have the same type $\mathbb{Q}_{\geq 0}^{\mathtt{X}}$, however we find it technically more convenient to work with reset points than with clock valuations. The reset-point semantics has already appeared in the literature on timed automata [49] and it is the foundation of the related model of timed-register automata [12].

Semantics of $\mathcal{A} \in$ NTA is given in terms of an LTS $[\![\mathcal{A}]\!] = (\Sigma, C, C_{\mathsf{ini}}, C_{\mathsf{fin}}, \rightarrow)$, where $C = L \times \mathbb{Q}_{\geq 0}^{\mathtt{X}} \times \mathbb{Q}_{\geq 0}$. A single *configuration* is a tuple (p, μ, τ_0) consisting of a control location $p \in L$, a reset-point assignment $\mu \in \mathbb{Q}_{\geq 0}^{\mathtt{X}}$, and a 'now' timestamp $\tau_0 \in \mathbb{Q}_{\geq 0}$ satisfying $\mu(\mathtt{x}) \leq \tau_0$ for all clocks $\mathtt{x} \in \mathtt{X}$. Intuitively, τ_0 is the last timestamp seen in the input and, for every clock $\mathtt{x}$, $\mu(\mathtt{x})$ stores the timestamp of the last reset of $\mathtt{x}$.

The set C_{ini} of *initial configurations* contains all tuples (p, μ, τ_0) where:

- $p \in L_{\mathsf{ini}}$
- $\tau_0 = 0$,
- $\mu(\mathtt{x}) = 0$ for all clocks $\mathtt{x}$.

Final configurations are simply defined as $C_{\mathsf{fin}} := L_{\mathsf{fin}} \times \mathbb{Q}_{\geq 0}^{\mathtt{X}} \times \mathbb{Q}_{\geq 0}$, without any further restriction on μ or τ_0.

For a set of clocks $\mathtt{Y} \subseteq \mathtt{X}$ and a timestamp $\xi \in \mathbb{Q}_{\geq 0}$, let $\mu[\mathtt{Y} \mapsto \xi]$ be the assignment which is ξ on $\mathtt{Y}$ and agrees with μ on $\mathtt{X} \setminus \mathtt{Y}$. A reset-point assignment μ together with τ_0 induces the *clock valuation* $\tau_0 - \mu$ defined as $(\tau_0 - \mu)(\mathtt{x}) = \tau_0 - \mu(\mathtt{x})$ for all clocks $\mathtt{x} \in \mathtt{X}$.

Every transition rule (2.6) induces a *transition* between configurations $(p, \mu, \tau_0) \xrightarrow{\sigma,\tau} (q, \nu, \tau)$ labelled by $(\sigma, \tau) \in \Sigma \times \mathbb{Q}_{\geq 0}$ whenever the following conditions hold:

- $\tau \geq \tau_0$,
- $\tau - \mu \models \varphi$, and
- $\nu = \mu[\mathtt{Y} \mapsto \tau]$.

Since there is no danger of confusion, we use $[\![\mathcal{A}]\!]$ to denote both the timed transition system above and its domain.

All notions defined for LTS $[\![\mathcal{A}]\!]$ in §2.2 remain valid and are inherited by timed automata. Henceforth, when talking of a 'run', an 'accepting run', or a 'language', etc., of $\mathcal{A}$, we will mean a corresponding notion for $[\![\mathcal{A}]\!]$. In the same vein, when writing $\mathcal{L}(\mathcal{A})$ or $\mathcal{L}^{\omega}(\mathcal{A}; c)$, we mean $\mathcal{L}([\![\mathcal{A}]\!])$ and $\mathcal{L}^{\omega}([\![\mathcal{A}]\!]; c)$, respectively. Additionally, for simplicity, we omit brackets whenever it does not cause any confusion, e.g., preferring $\mathcal{L}(\mathcal{A}; p, \mu, \tau_0)$ over $\mathcal{L}(\mathcal{A}; (p, \mu, \tau_0))$.

Observe that above semantics guarantees that $\mathcal{L}(\mathcal{A}; p, \mu, \tau_0) \subseteq \mathbb{T}_{\geq \tau_0}(\Sigma)$.

In an always resetting $\mathsf{NTA}_{\bullet,m}$, every reachable configuration (p, μ, τ_0) satisfies $\tau_0 \in \mu(\mathtt{X})$, where $\mu(\mathtt{X}) = \{\mu(\mathtt{x}) \mid \mathtt{x} \in \mathtt{X}\}$.

In a greedily resetting one:

1. (p, μ, τ_0) has *m-bounded span*, in the sense that $\mu(\mathtt{X}) \subseteq (\tau_0 - m, \tau_0]$, and moreover

2. any two clocks $\mathtt{x}, \mathtt{y}$ with integer difference $\mu(\mathtt{x}) - \mu(\mathtt{y}) \in \mathbb{Z}$ are actually equal $\mu(\mathtt{x}) = \mu(\mathtt{y})$.

Condition 2) follows from the fact that if $\mathtt{x}, \mathtt{y}$ have integer difference and $\mathtt{y}$ was reset last, then $\mathtt{x}$ was itself an integer when this happened, and in fact they were both reset together in a greedily resetting automaton.

2.5.4 Deterministic timed automata (DTA)

Deterministic timed automata (DTA) are a subclass of NTA. A timed automaton $\mathcal{A}$ is *deterministic* if its underlying LTS $[\![\mathcal{A}]\!]$ is so. This semantic requirement translates back to $\mathcal{A}$'s transition rules: $\mathcal{A}$ is deterministic if

- it has exactly one initial location, and
- for every two rules $(p, \sigma, \varphi, \mathtt{Y}, q), (p, \sigma', \varphi', \mathtt{Y}', q') \in \Delta$, if $\sigma = \sigma'$ and $[\![\varphi \wedge \varphi']\!] \neq \varnothing$ then $\mathtt{Y} = \mathtt{Y}'$ and $q = q'$. This condition guarantees that transition rules with common initial state and jointly satisfiable constraints induce coinciding transitions in $[\![\mathcal{A}]\!]$.

Properties above imply that $\mathcal{A}$ has at most one run over every timed word w.

The notion of a *total automaton* is slightly changed by the requirement of monotonicity for timed words. As timestamps can only increase while the word is being read, we say a timed automaton is total when for every location configuration $c = (p, \mu, \tau) \in C$ and timed letter $(\sigma, \tau_0) \in \Sigma \times \mathbb{Q}_{\geq 0}$ $(\tau_0 \geq \tau)$ there exists at least one transition $(p \xRightarrow{\sigma, \varphi, \mathtt{Y}} q)$ such that $\tau_0 - \mu \models \varphi$. As expected, total DTA has exactly one run over every timed word w.

Observe that a DTA can be easily transformed to an equivalent total one, with the following additional property: For every location $p \in L$ and $\sigma \in \Sigma$, the sets defined by clock constraints $\{[\![\varphi]\!] \mid \exists \mathtt{Y}, q \,.\, (p, \sigma, \varphi, \mathtt{Y}, q) \in \Delta\}$ form a partition of $\mathbb{Q}^{\mathtt{X}}_{\geq 0}$. We write DTA for the class of deterministic timed automata, and DTA_k, $\mathsf{DTA}_{\bullet,m}$, and $\mathsf{DTA}_{k,m}$ for the respective subclasses thereof. A timed language is called NTA language, DTA language, etc., if it is recognised by a timed automaton of the respective type.

Example 2.24. NTA language which is not a DTA language

This is a timed analogue of example 2.17. Let $\Sigma = \{\sigma\}$ be a unary alphabet. As an example of a timed language L recognised by an NTA_1, but not by any DTA, consider the set of non-negative timed words of the form $(\sigma, \tau_1) \cdots (\sigma, \tau_n)$ where $\tau_n - \tau_i = 1$ for some $1 \leq i < n$. The language L is recognised by the NTA_1 $\mathcal{A} = (\mathtt{X}, \Sigma, L, L_{\mathsf{ini}}, L_{\mathsf{fin}})$ with a single clock $\mathtt{X} = \{\mathtt{x}\}$ and three locations $L = \{p, q, r\}$, of which $L_{\mathsf{ini}} = \{p\}$ is initial and $L_{\mathsf{fin}} = \{r\}$ is final, and transition rules

$$(p \xRightarrow{\sigma, \mathsf{TRUE}, \varnothing} p) \qquad (p \xRightarrow{\sigma, \mathsf{TRUE}, \{\mathtt{x}\}} q) \qquad (q \xRightarrow{\sigma, \mathtt{x}<1, \varnothing} q) \qquad (q \xRightarrow{\sigma, \mathtt{x}=1, \varnothing} r).$$

Intuitively, in p the automaton waits until it guesses that the next input will be (σ, τ_i), at which point it moves to q by resetting the clock (and subsequently reading σ). From q, the automaton can accept by going to r only if exactly one time unit elapsed since

(σ, τ_i) was read. The language L is not recognised by any DTA since, intuitively, any deterministic acceptor needs to store unboundedly many timestamps τ_i.

Example 2.25. complement of language from 2.24 is NTA, but not DTA, too

The complement of L from example 2.24 can be recognised by an NTA with two clocks. Indeed, a timed word $(a, \tau_1) \cdots (a, \tau_n)$ is not in L if either of the following conditions hold:

1) its length n is at most 1, or
2) the total time elapsed between the first and the last letter is less than one time unit $\tau_n - \tau_1 < 1$, or
3) there is a position $1 \leq i < n$ such that $\tau_n - \tau_i > 1$ and $\tau_n - \tau_{i+1} < 1$.

It is easy to see that two clocks suffice to nondeterministically check the conditions above.

Since checking whether an NTA recognises a deterministic language is undecidable [46,93], there is no recursive bound on the number of clocks sufficient to deterministically recognise an NTA language (whenever possible). Thus NTA can be non-recursively more succinct than DTA w.r.t. number of clocks. However, in general such NTA recognise timed languages whose complement is not an NTA language. The next example shows a timed language which is both NTA and co-NTA recognisable, however the number of clocks of an equivalent DTA is at least exponential in the number of clocks of the NTA.

Example 2.26.

For $k \in \mathbb{N}$, let L_k be the set of strictly monotonic timed words $(a, t_1) \cdots (a, t_n)$ s.t. $t_n - t_i = 1$ where $i = n - 2^k$. The language L_k can be recognised by a $(2 \cdot k + 2)$-clock NTA $\mathcal{A}_k$ of polynomial size. There are clocks $\mathtt{x}_0, \mathtt{x}_1, \ldots, \mathtt{x}_k$ and $\mathtt{y}_0, \mathtt{y}_1, \ldots, \mathtt{y}_k$. Clock $\mathtt{x}_0$ is used to check strict monotonicity. Clock $\mathtt{y}_0$ is reset when the automaton guesses (a, t_i). The automaton additionally keeps track of the length of the remaining input. This is achieved by implementing a k-bit binary counter, where $\mathtt{x}_j = \mathtt{y}_j$ represents that the j-th bit is one. In order to set the j-th bit to one, the automaton resets $\mathtt{x}_j, \mathtt{y}_j$; to set it to zero, it resets only $\mathtt{x}_j$. This is correct thanks to strict monotonicity. At the end the automaton checks $\mathtt{y}_0 = 1$ and that the binary counter has value 2^k. Any deterministic automaton recognising L_k requires exponentially many clocks to store the last 2^k timestamps. The complement of L_k can be recognised by a $(2 \cdot k + 2)$-clock NTA of polynomial size. Indeed, a timed word is not in L_k if any of the following conditions hold: **1.** its length n is $\leq 2^k$, or **2.** $t_n - t_i < 1$ with $i = n - 2^k$, or **3.** $t_n - t_i > 1$ with $i = n - 2^k$. The automaton guesses which condition holds and uses a k-bit binary counter as above to check that position i has been guessed correctly.

2.6 Vector addition systems with states (VASS)

Vector addition systems [62] (VAS) are a widely accepted model of concurrency equivalent to Petri nets. Another equivalent model, called *vector addition systems with states*

(VASS) [58], extends VAS with a finite sets of control states. A VASS with d counters is said to be of *dimension* d; family of all such models being denoted as VASS_d.

Formally, a $\mathcal{V} \in \mathsf{VASS}_d$ is a tuple (Q, T), where $T \subseteq_{\mathsf{fin}} Q \times \mathbb{Z}^d \times Q$ is a finite set of transitions. Intuitively, each transition in T is assigned a vector describing how counter values change upon executing it. A *configuration* of a VASS is a pair $(q, v) \in Q \times \mathbb{N}^d$. Semantics of VASS is given by the transition system $[\![\mathcal{V}]\!] = (Q \times \mathbb{N}^d, \to)$, where $\to$ contains a transition $(q, v) \to (r, w)$ if, and only if there exists a transition rule $(q, \delta, r) \in T$ such that $v + \delta = w$.

Problems concerning VASS, which are frequently studied in literature include

- reachability—given $c, c' \in [\![\mathcal{V}]\!]$, does $c \to^* c'$ hold?
- coverability—given two configurations $c = \in [\![\mathcal{V}]\!]$ and $c' = (r, w) \in [\![\mathcal{V}]\!]$, does there exist $\delta \in \mathbb{N}^d$ such that $c \to^* (r, w + \delta)$?.
- boundedness—given $c \in [\![\mathcal{V}]\!]$, is the set of reachable configurations $\{c' \in [\![\mathcal{V}]\!] \mid c \to^* c'\}$ finite?

In this thesis, we exclusively focus on VASS of dimension 2, all subsequent definitions related to VASS reflect that.

We refer to elements of $\mathbb{Z}^2$ as *vectors*. *Non-negative* vectors are elements of $\mathbb{N}^2$, and *positive* vectors are elements of $\mathbb{Z}^2_{>0}$. A vector with only non-negative coordinates and at least one positive coordinate is called *semi-positive*; it is either positive, or *vertical* of the form $(0, a)$, or *horizontal* of the form $(a, 0)$, for $a \in \mathbb{Z}_{>0}$. Additionally, by $(0, 0)$ we denote the vector $(0, 0)$.

A 2-dimensional *vector addition system with states* (VASS_2) $\mathcal{V}$ consists of a finite set of control states Q and a finite set of transitions $T \subseteq Q \times \mathbb{Z}^2 \times Q$. We refer to the vector v as the *effect of a transition* (p, v, q). A *path* in $\mathcal{V}$ from control state p to q is a sequence of transitions $\pi = (q_0, v_1, q_1), (q_1, v_2, q_2), \ldots, (q_{n-1}, v_n, q_n) \in T^*$ where $p = q_0$ and $q = q_n$; it is called a *cycle* whenever the starting and ending control states coincide ($q_0 = q_n$). The *effect of a path* is defined as $\mathsf{eff}(\pi) = v_1 + \ldots + v_n \in \mathbb{Z}^2$, and its *length* is n. A cycle is called non-negative, semi-positive or positive, if its effect is so.

A *configuration* of $\mathcal{V}$ is an element of $\mathrm{Conf} = Q \times \mathbb{N}^2$. A transition $t = (p, v, q)$ is *enabled* in a configuration $c = (p', u)$ if $p = p'$ and $u + v \in \mathbb{N}^2$. Analogously, a path π is enabled in a configuration $c = (p', u)$ if $q_0 = p'$ and $u_i = u + v_1 + \ldots + v_i \in \mathbb{N}^2$ for every i. In such case we say that π induces a *run* of the form

$$\rho = (c_0, t_1, c_1), (c_1, t_2, c_2), \ldots, (c_{n-1}, t_n, c_n) \in (\mathrm{Conf} \times T \times \mathrm{Conf})^*$$

with intermediate configurations $c_i = (q_i, u_i)$, from the *source* configuration $\mathsf{src}(\rho) = c_0$ to the *target* one $\mathsf{trg}(\rho) = c_n$. If the source configuration c_0 is clear from the context, we do not distinguish between a path enabled in c_0 and a run with source c_0, and simply say that the path *is* the run. A $(0, 0)$-*run* is a run whose source and target are $(0, 0)$-*configurations*, i.e., a configuration whose vector is $(0, 0)$.

We will sometimes relax the non-negativeness requirement on some coordinates: For $j \in \{1, 2\}$, we say that a path π is $\{j\}$-*enabled* in a configuration $c = (p', u)$ if $q_0 = p'$ and $(u + v_1 + \ldots + v_i)[j] \in \mathbb{N}$ for every i. We also say that π is $\varnothing$-*enabled* in c if just $q_0 = p'$.

The *reversal* of a VASS_2 $\mathcal{V} = (Q, T)$, denoted $\mathsf{rev}(\mathcal{V})$, is a VASS_2 with the same control states and with transitions $\{(q, -v, p) \mid (p, v, q) \in T\}$. We sometimes speak of the reversal $\mathsf{rev}(\rho)$ of a run ρ of $\mathcal{V}$, implicitly meaning a run in the reversal of $\mathcal{V}$.

As the *norm* of $v = (v_1, v_2) \in \mathbb{R}^2$, we take the largest of absolute values of v_1 and v_2, $\|v\| := \max\{|v_1|, |v_2|\}$. By the norm of a configuration $c = (q, v)$ we mean the norm of its vector v, and by the norm $\|\mathcal{V}\|$ of a VASS_2 $\mathcal{V}$ we mean the largest among norms of effects of transitions.

2.7 Computational problems

The computational problems of interest in this thesis have already been outlined in §1.2, which discusses their context in greater detail and presents our contributions. This section complements it by providing concise and fully formal definitions of these problems.

2.7.1 Synthesis problem and synthesis game

Church's synthesis problem

The synthesis problem proposed by Alonso Church in his work from 1957 became one of the cornerstones of program synthesis. Intuitively, the task is to produce a *finite-state controller*—an automaton with an output—which reacts to the *input* symbols it receives in such a way as to always remain within a given *specification*.

The original formulation of the problem is as follows, with specification in terms of monadic second order logic (MSO).

Problem 2.27. original Church's synthesis

Input:	specification in form of a formula φ of $\mathsf{MSO}(\mathbb{N}, <)$ (a.k.a. MLO)
Output:	▶ if it exists, return a finite automaton with output $\mathcal{S}$ which for every input word w outputs $\mathcal{S}(w)$, such that the input-output pair $(w, \mathcal{S}(w))$ satisfies φ, ▶ else return Nil.

We omit the details of how infinite words are handled by the formula. As the order $<$ is MSO-definable by the means of the successor function, above $\mathsf{MSO}(\mathbb{N}, <)$ can be replaced with $\mathsf{MSO}(\mathbb{N}, +1)$ (a.k.a. S1S). We will not delve into logic, as it is not within the area of interest of this dissertation. Instead, we use the classical result of Büchi which relates $\mathsf{MSO}(\mathbb{N}, +1)$ with ω-regular languages. It is easy to prove that for every $\mathcal{W} \in \mathsf{NFA}$ there exists a $\mathsf{MSO}(\mathbb{N}, +1)$ formula $\varphi_{\mathcal{W}}$ which defines the same language of infinite words:

$$\mathcal{L}^{\omega}(\varphi_{\mathcal{W}}) = \mathcal{L}^{\omega}(\mathcal{W}).$$

In his paper [87], Büchi has shown that the opposite implication also holds

Theorem 2.28. [87, corollary of lemma 10 and theorem 1]

For every $\mathsf{MSO}(\mathbb{N}, +1)$ formula φ, there exists an NFA with the same language of infinite words.

Given the above theorem, Church's problem can be equivalently formulated as

Problem 2.29. reformulated Church's synthesis

INPUT: specification in form of an NFA $\mathcal{W}$ over the finite alphabet $A \times B$

OUTPUT: ▶ if it exists, return $\mathcal{S} \in$ DFA(A, B) which for every infinite input word $v = v_1 v_2 v_3 \cdots \in A^\omega$ produces an output word $\mathcal{S}(v) = w_1 w_2 w_3 \cdots \in B^\omega$ such that

$$(v_1, w_1)(v_2, w_2)(v_3, w_3) \cdots \in \mathcal{L}^\omega(\mathcal{W}),$$

▶ else return NIL.

Büchi-Landweber theorem

Twelve years after the Church's problem was stated, Büchi and Landweber presented its solution in their paper [15].

Theorem 2.30. [15, theorem 6.]

The Church's synthesis problem is computable.

Synthesis games

Already in the paper solving the Church's problem, the authors provide an alternative formulation of the synthesis problem. It is rephrased as a question of existence of a winning finite-memory strategy in an appropriately tailored game. They provide a two-player zero-sum[8] game of infinite duration, which matches the framework of Gale-Stewart ω-games [50]. Additionally, one of the results from [15] shows that—unlike in the general case of ω-games—its winner is always determined.

Definition 2.31. synthesis game

The *synthesis game* $G_{A,B}(\mathcal{W})$ is specified by its winning condition—the language $W = \mathcal{L}^\omega(\mathcal{W})$ of some automaton $\mathcal{W} \in$ NRA. There are two players, Alice and Bob, who alternately play symbols from finite alphabets of actions A and B. In the i-th turn, Alice plays some a_i, and her opponent replies with b_i; both can choose their moves based on the history of all past turns (perfect information). They play indefinitely, thus creating an infinite word

$$\pi = (a_1, b_1)(a_2, b_2)(a_3, b_3) \cdots \in (A \times B)^\omega \tag{2.7}$$

called a *play*. Bob wins if the play satisfies the winning condition: $\pi \in W$.

In some cases, it is more convenient to drop the pair brackets and write the play π as $a_1 b_1 a_2 b_2 a_3 b_3 \cdots \in (A \cup B)^\omega$. When there is no danger of confusion, we will be silently switching between these notations. The set of all possible plays of $G_{A,B}(W)$ is denoted as $\mathsf{Plays}(G_{A,B}(W))$.

[8] A game with a property that when one player wins, the other one loses.

Strategies and controllers

The game $G_{A,B}(\mathcal{W})$ relates to the synthesis problem in the following way: Alice provides the input word, while Bob reacts to its every symbol, in a letter-by-letter fashion, just as an automaton with output would do. A *strategy* for Bob is any function of type $A^+ \to B$, which, based on the whole history of letters played by Alice so far, determines the action $b \in B$ which Bob should play in response. We say that a play π as in eq. (2.7) is *conformant* with a strategy $f : A^+ \to B$ (or f-conformant) whenever $b_n = f(a_1 a_2 \dots a_n)$ for every $n \in \mathbb{N}$. A strategy f is *winning*, if every f-conformant play is winning for Bob.

Observe that any strategy can be naturally represented by an (infinite) deterministic LTS with output[9]. Conversely, every $\mathcal{S} \in \mathsf{DLTS}(A, B)$ corresponds to some strategy, denoted $\mathsf{Strategy}(\mathcal{S})$, mapping every $v \in A^+$ to the last letter of $\mathcal{S}(v)$.

When the strategy can be represented by $\mathsf{DFA}(A, B)$, we call it a *finite-memory strategy*. Henceforth, deterministic automata with output in the context of games are also called *controllers*. We say that the controller $\mathcal{S}$ is winning whenever $\mathsf{Strategy}(\mathcal{S})$ is, and that a play is $\mathcal{S}$-conformant whenever it is $\mathsf{Strategy}(\mathcal{S})$-conformant.

Similarly, we can define the strategy for Alice as any function $g : B^* \to A$. Although it is of little relevance to this thesis, we give its definition for the sake of completeness. A technical difference between the two analogous notions is that the function is specified also for the empty word ε (hence the domain is B^* and not B^+). It represents the initial action of Alice, which has to be played in absence of any previous actions of Bob.

A play π is g-conformant whenever $a_n = g(b_1 b_2 \dots b_{n-1})$ for every $n \in \mathbb{N}_{\geq 1}$. Notion of a winning strategy is defined similarly to one of Bob.

Problem 2.32. Church's synthesis—game-theoretic approach

INPUT: winning condition given by an NFA $\mathcal{W}$ over the finite alphabet $A \times B$
OUTPUT:
- ▶ if it exists, return a winning controller $\mathcal{S} \in \mathsf{DFA}(A, B)$ for Bob.
- ▶ else return NIL.

2.7.2 Deterministic separability problems

Separation

We say that a set S *separates* A from B whenever

$$A \subseteq S \wedge B \cap S = \varnothing.$$

The general *language separability problem* below is parametrised with classes X, Y of models of computation, and possibly with some additional parameters.

Problem 2.33. general language separability problem

INPUT: $\mathcal{A}, \mathcal{B} \in \boxed{X}$
OUTPUT:
- ▶ $\mathcal{S} \in \boxed{Y}$ such that $\mathcal{L}(\mathcal{S})$ separates $\mathcal{L}(\mathcal{A})$ from $\mathcal{L}(\mathcal{B})$, if it exists;
- ▶ NIL otherwise.

[9] We can construct an infinite *tree*, whose branches correspond to words from A^+ played by Alice.

Deterministic separability problems

This thesis considers it instantiated for both timed and register automata. More precisely, we show decidability of its three variants below, in all of which nondeterministic models are to be separated by a deterministic device.

Problem 2.34. k-clock m-constrained deterministic separability of NTA

Separability problem 2.33 parametrised by $k, m \in \mathbb{N}$, with $X :=$ NTA and $Y :=$ $\mathsf{DTA}_{k,m}$.

Solved by theorem 4.3 on page 71.

Problem 2.35. k-clock deterministic separability of NTA

Separability problem 2.33 parametrised by $k \in \mathbb{N}$, with $X :=$ NTA and $Y :=$ DTA_k.

Solved by theorem 4.2 on page 71.

Problem 2.36. deterministic separability of NTA

Separability problem 2.33 with $X :=$ NTA and $Y :=$ DTA.

Open.

Problem 2.37. k-clock deterministic separability of NRA

Separability problem 2.33 parametrised by $k \in \mathbb{N}$, with $X :=$ NRA and $Y :=$ DRA_k.

Solved by theorem 4.1 on page 71.

Problem 2.38. determinisic separability of NRA

Separability problem 2.33 with $X :=$ NRA and $Y :=$ DRA.

Open.

2.7.3 Deterministic membership problems

As with the separability problem, the generic version of the *membership problem* is parametrised with two classes X, Y of models of computation.

Problem 2.39. general membership problem

Input: $\mathcal{A} \in X$
Output:
- ▶ $\mathcal{B} \in Y$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$, if it exists;
- ▶ Nil otherwise.

Deterministic membership problems

We consider this problem for both timed and register automata, with X and Y being chosen as 1-register/clock nondeterministic and deterministic models, respectively.

Problem 2.40. DRA_k membership of NRA_1

Membership problem 2.39 parametrised by $k \in \mathbb{N}$ with $X \coloneqq \mathsf{NRA}_1$ and $Y \coloneqq \mathsf{DRA}_k$.

Solved by theorem 7.1 on page 95.

Problem 2.41. $\mathsf{DTA}_{k,m}$ membership of NTA_1

Membership problem 2.39 parametrised by $k, m \in \mathbb{N}$, with $X \coloneqq \mathsf{NTA}_1$ and $Y \coloneqq \mathsf{DTA}_{k,m}$.

Solved by theorem 7.29 on page 108.

Problem 2.42. DTA_k membership of NTA_1

Membership problem 2.39 parametrised by $k \in \mathbb{N}$ with $X \coloneqq \mathsf{NTA}_1$ and $Y \coloneqq \mathsf{DTA}_k$.

Solved by theorem 7.28 on page 108.

Chapter 3

Generalised synthesis problems and games

This chapter introduces our proposed extensions of the framework of ω-synthesis games (from §2.7.1), adapting it to the setting of register and timed automata. As our register synthesis games and timed synthesis games share a lot of similarities, we shall start with an abstract synthesis game parametrised by an arbitrary device $\mathcal{W}$. This approach, in addition to avoiding some repetitions, allows us to better highlight the subtle differences between games, when the abstract definition is later materialised in subsequent sections.

3.1 Generalised deterministic LTS synthesis game

Church's synthesis generalised to infinite alphabets

Below, there is an abstract generalisation of the Church's synthesis problem, which adds support for an infinite input alphabet.

Problem 3.1. generalised Church's synthesis

Input: specification given by a model of computation $\mathcal{W}$, which induces $[\![\mathcal{W}]\!] \in$ LTS$(A \times B \times \mathbb{X})$ (where A, B are finite, while $\mathbb{X}$ does not have to)

Output: ▶ if it exists, return a model $\mathcal{S}$ inducing $[\![\mathcal{S}]\!] \in$ LTS$(A \times \mathbb{X}, B)$, which for every infinite input word $v = (v_1, \chi_1)(v_2, \chi_2)(v_3, \chi_3) \cdots \in (A \times \mathbb{X})^\omega$ produces an output word $[\![\mathcal{S}]\!](v) = w_1 w_2 w_3 \cdots \in B^\omega$ such that

$$(v_1, w_1, \chi_1)(v_2, w_2, \chi_2)(v_3, w_3, \chi_3) \ldots \notin \mathcal{L}^\omega(\mathcal{W}),$$

▶ else return Nil.

For the above problem to be considered from a practical perspective and enable algorithmic manipulation, both LTS have to be finitely represented through some models of computation $\mathcal{W}$ and $\mathcal{S}$. We will consider variants of it, in which these LTS are given as the semantics of either register or timed automata.

Important difference in winning condition

In the original Church's problem (problem 2.29), the word $(v_1, w_1)(v_2, w_2)(v_3, w_3)\dots$ combining the input and output was required to belong to the language $\mathcal{L}^\omega(\mathcal{W})$. This is not the case in the generalised game we propose. It has a simple explanation. The original problem features a winning condition which is a ω-regular language, and these are effectively closed under complement. Therefore, in problem 2.29, it is not important if the condition specifies *desired* or *undesired* behaviours of controllers. However, once we move to the setting of register and timed automata, whose languages are no longer closed under the complement, the way the problem is stated influences its computability.

Generalised synthesis game

As in case of the base Church's synthesis, we introduce a *generalised synthesis game* which allows us to formulate problem 3.1 in a game-theoretic language.

Definition 3.2. generalised synthesis game

The *synthesis game* $G^{\mathbb{X}}_{A,B}(\mathcal{W})$ is specified by its winning condition—the language $W = \mathcal{L}^\omega(\mathcal{W}) \subseteq (A \times B \times \mathbb{X})^\omega$ of some model $\mathcal{W}$. There are two players, Alice and Bob, who alternately play symbols from alphabets of actions $A \times \mathbb{X}$ and B, sets A, B being finite, and $\mathbb{X}$ possibly not. In the i-th turn, Alice plays some $(a_i, \chi_i) \in A \times \mathbb{X}$, and her opponent replies with $b_i \in B$; both can choose their moves based on the history of all past turns (perfect information). They play infinitely, thus creating an infinite play

$$\pi = (a_1, b_1, \chi_1)(a_2, b_2, \chi_2)(a_3, b_3, \chi_3)\dots \in (A \times B \times \mathbb{X})^\omega \tag{3.1}$$

Alice wins if the play satisfies the winning condition: $\pi \in W$.

We define two *auxiliary projections*, $\mathsf{proj_A} : A \times B \times \mathbb{X} \to A \cdot \mathbb{X}$ and $\mathsf{proj_B} : A \times B \times \mathbb{X} \to B$, to help us extract the subsequences of plays produced only by Alice and Bob, respectively:

$$\mathsf{proj_A}(a, b, \chi) := (a, \chi)$$
$$\mathsf{proj_B}(a, b, \chi) := b.$$

Both functions are extended homomorphically to finite and infinite words, and also to languages. More precisely

$$\mathsf{proj_A}(c_1 c_2 \dots) = \mathsf{proj_A}(c_1)\mathsf{proj_A}(c_2)\cdots \qquad \text{for } c_\bullet \in A \times B \times \mathbb{X}$$
$$\mathsf{proj_A}(L) = \{\mathsf{proj_A}(w) \mid w \in L\} \qquad \text{for } L \subseteq (A \times B \times \mathbb{X})^\omega$$

case of $\mathsf{proj_B}$ being handled analogously.

Similarly to §2.7.1, a *strategy* for Bob is any function f of type $(A \times \mathbb{X})^+ \to B$, the only difference being that now the set of actions $A \times \mathbb{X}$ is allowed to be infinite. Again, based on the history of Alice's actions, a strategy yields the symbol $b \in B$ which Bob should use in the current turn. A play πas in eq. (3.1) is said to be *f-conformant* whenever $b_n = f((a_1, \alpha_1)(a_2, \alpha_2), \dots, (a_n, \alpha_n))$ for every $n \in \mathbb{N}_{\geq 1}$. Again, as in the finite-alphabet case, strategies can be represented by infinite $\textsc{dlts}(A \times \mathbb{X}, B)$, and vice versa, every $\mathcal{T} \in \textsc{dlts}(A \times \mathbb{X}, B)$ with matching alphabets corresponds to a strategy, denoted by $\mathsf{Strategy}(\mathcal{T})$, outputting the last letter of $\mathcal{S}(w)$ for every w.

A *controller* for Bob in $G^{\mathbb{X}}_{A,B}(\mathcal{W})$ is any model of computation $\mathcal{S}$, such that $[\![\mathcal{S}]\!] \in$ DLTS$(A \times \mathbb{X}, B)$. We say that it is winning whenever $\mathsf{Strategy}([\![\mathcal{S}]\!])$ is; and that π is $\mathcal{S}$-conformant, whenever it is $\mathsf{Strategy}([\![\mathcal{S}]\!])$-conformant.

Observe that if a play π as in eq. (3.1) is $\mathcal{S}$-conformant, it corresponds to exactly one run σ of $[\![\mathcal{S}]\!]$ of the form

$$\mathsf{runToPlay}(\pi) = c_0 \xrightarrow{(a_1,\xi_1)/b_1} c_1 \xrightarrow{(a_2,\xi_2)/b_2} c_2 \xrightarrow{(a_3,\xi_3)/b_3} \cdots \in \mathsf{Runs}([\![\mathcal{S}]\!]),$$

where $c_\bullet$ are configurations of $[\![\mathcal{S}]\!]$. We write $\pi = \mathsf{runToPlay}(\sigma)$.

Problem 3.3. generalised Church's synthesis—game-theoretic approach

INPUT: winning condition given by a model $\mathcal{W}$, which induces LTS $[\![\mathcal{W}]\!]$ over the alphabet $A \times B \times \mathbb{X}$

OUTPUT:
- ▶ if it exists, return a controller $\mathcal{S}$ (with input alphabet $A \times \mathbb{X}$ and output alphabet B), such that $[\![\mathcal{S}]\!]$ is winning for Bob.
- ▶ else return NIL.

3.2 Register synthesis game

Register synthesis game is the first of two synthesis games we investigate in this dissertation. Connected to it is the *register synthesis problem*, which arises as a generalisation of the Church's problem to the setting of register automata—a restricted form of problem 3.1 parametrised by structure $\mathbb{A}$ of atoms, and with $\mathcal{W}$ and $\mathcal{S}$ being NRA$^{\mathrm{g}}$ and DRA with output, respectively.

Definition 3.4. register synthesis game

The *register synthesis game* $G^{\mathbb{A}}_{A,B}(\mathcal{W})$ is a game $G^{\mathbb{X}}_{A,B}(\mathcal{W})$, where:

- ▶ the infinite component of the alphabet is the set of atoms: $\mathbb{X} := \mathbb{A}$,
- ▶ the winning condition is given by $\mathcal{W} \in$ NRA$^{\mathrm{g}}$ over the alphabet $A \times B$,
- ▶ Alice plays a letter from a finite alphabet A paired with an atom from $\mathbb{A}$,
- ▶ Bob has a finite set of actions B,
- ▶ Alice wins if, and only if, $\pi \in \mathcal{L}^\omega(\mathcal{W})$.

A *controller* for Bob in $G^{\mathbb{A}}_{A,B}(W)$ is any timed automaton from DRA(A, B). We also consider a subclass of controllers parametrised in the number of used registers: a *k-register controller* for Bob is any $\mathcal{A} \in$ DRA$_k(A, B)$.

Problem 3.5. register synthesis problem

Parametrised by a *homogeneous* structure $\mathbb{A}$ of atoms over finite relational vocabulary.

INPUT: winning condition—register automaton $\mathcal{W} \in \mathsf{NRA}^{\mathsf{g}}$ over the alphabet $A \times B$
OUTPUT:
- ▶ if it exists, return a winning $\mathcal{S} \in \mathsf{DRA}(A, B)$.
- ▶ else return NIL.

Its associated decision variant is undecidable—theorem 8.4 on page 119.

Problem 3.6. *k-register synthesis problem*

Problem 3.5 parametrised by $k \in \mathbb{N}$, and additionally requiring $\mathcal{S}$ to belong to $\mathsf{DRA}_k(A, B)$.

Solved by theorem 5.1 on page 75.

3.3 Timed synthesis game

Timed synthesis game is closely resembling the register synthesis game we introduced, introducing only minor technical assumption about the controllers.

Definition 3.7. *timed synthesis game*

The *timed synthesis game* $G^{\mathbb{T}}_{A,B}(\mathcal{W})$ is a game $G^{\mathbb{X}}_{A,B}(\mathcal{W})$, where:

- ▶ the infinite component of the alphabet is the set of timestamps: $\mathbb{X} := \mathbb{Q}$,
- ▶ the winning condition is given by $\mathcal{W} \in \mathsf{NTA}$ over the alphabet $A \times B$,
- ▶ Alice plays a letter from a finite alphabet A paired with a timestamp from $\mathbb{Q}$,
- ▶ Bob has a finite set of actions B,
- ▶ Alice wins if, and only if, $\pi \in \mathcal{L}^{\omega}(\mathcal{W})$.

In the timed case, a *controller* for Bob in $G^{\mathbb{T}}_{A,B}(W)$ is a $\mathsf{DTA}(A, B)$. We also consider subclass of controllers:

- ▶ a *k-clock timed controller* for Bob is a timed controller, which is a DTA_k with output,
- ▶ a *k-clock m-constrained timed controller* for Bob is a timed controller, which is a $\mathsf{DTA}_{k,m}$ with output.

For technical reasons, it is convenient to assume that timed controllers are *regionised*, i.e., that they feature only maximal clock constraints. It is not a problematic assumption, because every controller can be easily brought to regionised form. Indeed, its every transition rule $(p \xRightarrow{\sigma,\varphi,\mathtt{Y}} q)$ can be replaced by a number of rules $(p \xRightarrow{\sigma,\varphi_i,\mathtt{Y}} q)$ labelled with maximal clock constraints φ_i. Since φ_i correspond to elements of $\mathsf{Reg}(k, m)$, we can safely change the type of the transition relation δ to

$$\delta \subseteq L \times A \times \mathsf{Reg}(k, m) \times B \times \mathcal{P}(\mathtt{Y}) \times L,$$

the change being only superficial and not affecting the semantics.

Problem 3.8. *timed synthesis problem*

INPUT: winning condition—timed automaton $\mathcal{W} \in \mathsf{NTA}$ over the alphabet $A \times B$

OUTPUT:
- ▶ if it exists, return a winning controller $\mathcal{S} \in \mathsf{DTA}(A, B)$, such that $\mathsf{Strategy}(\mathcal{S})$ is winning for Bob.
- ▶ else return NIL.

Its associated decision variant is undecidable—theorem 8.2 on page 119.

Problem 3.9. *k-clock timed synthesis problem*

Problem 3.8 parametrised by $k \in \mathbb{N}$, and additionally requiring $\mathcal{S}$ to belong to $\mathsf{DTA}_k(A, B)$.

Solved by theorem 6.2 on page 81.

Problem 3.10. *k-clock m-constrained timed synthesis problem*

Problem 3.8 parametrised by $k, m \in \mathbb{N}$, and additionally requiring $\mathcal{S}$ to belong to $\mathsf{DTA}_{k,m}(A, B)$.

Solved by theorem 6.1 on page 81.

Chapter 4

Synthesis game for solving deterministic separability

Before moving on to the technical part, in which we solve the synthesis problems posed in §3, we first demonstrate their practical applicability. As it turns out, the separability problems 2.34, 2.35 and 2.37 can all be solved with a simple reduction to an appropriate variant of synthesis problem:

Theorem 4.1. separability by DRA_k

For $k \in \mathbb{N}$, the k-register deterministic separability problem for NRA is decidable.

Theorem 4.2. separability by DTA_k

For $k \in \mathbb{N}$, the k-clock deterministic separability problem for NTA are decidable.

Theorem 4.3. separability by $\mathsf{DTA}_{k,m}$

For $k, m \in \mathbb{N}$, the k-clock m-constrained separability problem for NTA is decidable.

It is enough to define a single instance of the general synthesis game—we call it *separation game*—and show that Bob's winning strategy corresponds directly to a separator. This is, one might say, a rather unusual case: typically, the results for timed and register automata, despite having analogous formulations, often differ in their proofs in subtle but non-negligible technical issues.

4.1 Motivating examples

We begin with a motivating example of non-separable timed languages.

Example 4.4.

Recall examples 2.24 and 2.25. Both the NTA language L from the former example and its complement from the latter are not DRA. It follows that they cannot be deterministically separable.

Moreover, a deterministic separator, when it exists, may need exponentially many clocks.

Example 4.5.

We have seen in example 2.26 an $O(k)$-clock NTA language such that 1) its complement is also an $O(k)$-clock NTA language, and 2) any DTA recognising it requires 2^k clocks. Thus, a deterministic separator may need exponentially many clocks in the size of the input NTA.

We do not know if there is an upper bound on the number of clocks/registers of the deterministic separator, if it exists. The separability game from the next section does not yield it, as the unconstrained synthesis problem is undecidable(theorems 8.2 and 8.4).

4.2 Reduction to generalised synthesis problem

Intuition

In order to solve all three separability problems at once, we define a generalised synthesis game which essentially characterizes the concept of *separability*. The core abstract result here is to show that Bob's winning strategies of type $(\Sigma \times \mathbb{X})^+ \to \{\textsf{ACC}, \textsf{REJ}\}$ are—save the case of empty word—characteristic functions of separators.

Separation game

Let $\mathcal{A}, \mathcal{B}$ be two models of computation such that $[\![\mathcal{A}]\!], [\![\mathcal{B}]\!] \in \textsf{LTS}(\Sigma \times \mathcal{X})$, Σ being a finite alphabet. A *separation game* $G^{\textsf{sep}}_{\Sigma\times\mathbb{X}}(\mathcal{A}, \mathcal{B})$ is a generalised synthesis game $G^{\mathbb{X}}_{\Sigma,B}(W)$ with two-element set $B = \{\textsf{ACC}, \textsf{REJ}\}$ of Bob's actions and with the winning condition of Alice being $W = \textsc{Reach}(V)$ for

$$
\begin{aligned}
V = (& \\
& \mathsf{proj}_{\mathsf{A}}^{-1}(\mathcal{L}(\mathcal{A})) \cap \mathsf{proj}_{\mathsf{B}}^{-1}(B^* \cdot \textsf{REJ}) \cup \qquad && \text{(element in } \mathcal{L}(\mathcal{A}) \text{ rejected)} \\
& \mathsf{proj}_{\mathsf{A}}^{-1}(\mathcal{L}(\mathcal{B})) \cap \mathsf{proj}_{\mathsf{B}}^{-1}(B^* \cdot \textsf{ACC}) && \text{(element in } \mathcal{L}(\mathcal{B}) \text{ accepted)} \\
). &
\end{aligned}
$$

Intuitively, Bob in every turn needs to classify the word $w \in (\Sigma \times \mathbb{X})^+$ Alice has played so far. If it belongs to $\mathcal{L}(\mathcal{A})$, his answer needs to be ACC, and when $w \in \mathcal{L}(\mathcal{B})$, REJ is required. In other cases, the answer may be arbitrary. Alice's winning condition describes two situations when Bob incorrectly classifies w.

Lemma 4.6.

For a function $f : (\Sigma \times \mathbb{X})^* \to B$ such that $f(\varepsilon) = \textsf{ACC} \iff \varepsilon \in \mathcal{L}(\mathcal{A})$, t.f.a.e.:

1. f restricted to $(\Sigma \times \mathbb{X})^+$ is a winning strategy for Bob in $G^{\textsf{sep}}_{\Sigma\times\mathbb{X}}(\mathcal{A}, \mathcal{B})$,
2. f is a characteristic function of a language S separating $\mathcal{L}(\mathcal{A})$ from $\mathcal{L}(\mathcal{B})$.

Proof of lemma 4.6.

Fix f satisfying the premise of lemma 4.6. Let $f' = f_{|(\Sigma\times\mathbb{X})^+}$.

$(1) \Rightarrow (2)$

Suppose f' is a winning strategy of Bob. We need to show that $S = f^{-1}(\{\textsf{ACC}\})$ separates $\mathcal{L}(\mathcal{A})$ from $\mathcal{L}(\mathcal{B})$, i.e., that for any $w \in (\Sigma \times \mathbb{X})^*$

$$w \in \mathcal{L}(\mathcal{A}) \implies w \in S \;\wedge\; w \in \mathcal{L}(\mathcal{B}) \implies w \notin S. \tag{4.1}$$

Take any $w \in (\Sigma \times \mathbb{X})^*$. If $w = \varepsilon$, the separability condition is trivially met. Otherwise, fix any f'-conformant play π of $G^{\textsf{sep}}_{\Sigma\times\mathbb{X}}(\mathcal{A}, \mathcal{B})$, with prefix $\pi' \sqsubset \pi$ satisfying $\textsf{proj}_{\textsf{A}}(\pi') = w$, i.e., one where Alice started by playing w. Since the strategy is winning, we conclude that $\pi \notin W$, and in particular $\pi' \notin V$. Therefore, w satisfies

$$\neg\big(w \in \mathcal{L}(\mathcal{A}) \wedge w \notin S \;\vee\; w \in \mathcal{L}(\mathcal{B}) \wedge w \in S\big)$$

which is equivalent to the required separability condition in eq. (4.1) above.

$(1) \Leftarrow (2)$

Suppose f separates $\mathcal{L}(\mathcal{A})$ from $\mathcal{L}(\mathcal{B})$. Therefore, for any $w \in (\Sigma \times \mathbb{X})^*$ it satisfies eq. (4.1). We conclude that for any f'-conformant run π, its every finite prefix $\pi' \sqsubset \pi$ does not belong to V, and thus $\pi \notin W$, which confirms that strategy f' is winning for Bob. ☐

Proof of theorem 4.1.

It is enough to show that problem 2.37 (k-register det. separability) reduces to k-register synthesis—decidable due to theorem 5.1. Fix $k \in \mathbb{N}$ and $\mathcal{A}, \mathcal{B} \in \textsf{NRA}^{\textsf{g}}$. Observe that for this choice of parameters $G = G^{\textsf{sep}}_{\Sigma\times\mathbb{X}}(\mathcal{A}, \mathcal{B})$ becomes a register synthesis game. Indeed, its winning condition W is recognised by $\textsf{NRA}^{\textsf{g}}$, as the class of $\textsf{NRA}^{\textsf{g}}$ languages is closed under inverse homomorphic images, intersections, and unions. We need to show there exists a winning controller$\mathcal{S} \in \textsf{DRA}_k(\Sigma, B)$ for Bob in G if, and only if, there exists a $\textsf{DRA}_k$ separating $\mathcal{L}(\mathcal{A})$ from $\mathcal{L}(\mathcal{B})$.

The '⇒' implication

Assume that $\mathcal{S} = (\mathtt{X}, \Sigma, L, L_{\textsf{ini}}, \_, \delta) \in \textsf{DRA}_k(\Sigma, B)$ is a winning controller. From lemma 4.6, $\textsf{Strategy}(\mathcal{S})$ is the characteristic function of a separating language L (modulo empty word ε). It is recognised by $\mathcal{S}' \in \textsf{DRA}_k$ easily obtained from $\mathcal{S}$. Conceptually, the construction amounts to changing the acceptance condition from 'output $\textsf{ACC}$ as the last letter of $\mathcal{S}(w)$' to 'visit accepting state after reading w'. Consider $\mathcal{S}' = (\mathtt{X}, \Sigma, L \times \{\textsf{ACC}, \textsf{REJ}\}, L_{\textsf{ini}} \times \{b_0\}, L \times \{\textsf{ACC}\}, \delta')$, where for any transition rule $(p \xRightarrow{\sigma,b,\varphi} q) \in \delta$, relation δ' contains

$$((p, \textsf{ACC}) \xRightarrow{\sigma,\varphi} (q, b)) \qquad\qquad ((p, \textsf{REJ}) \xRightarrow{\sigma,\varphi} (q, b)).$$

Above, $b_0 = \textsf{ACC}$ if, and only if $\varepsilon \in \mathcal{L}(\mathcal{A})$. It is easy to see that $\mathcal{L}(\mathcal{S}') = L$.

The '$\Leftarrow$' implication

A similar construction allows us to define a winning controller, based on a separator $\mathcal{S}' \in \mathsf{DRA}_k$. We only need to correctly assign output labels to transition rules: those ending in an accepting state are labeled with ACC, while the others—with REJ. Constructed $\mathsf{DRA}_k(\Sigma, B)$ is a winning controller by lemma 4.6. $\square$

Proof of theorems 4.2 and 4.3.

Proofs are analogous to the proof of theorem 4.1, with reductions to suitable synthesis problems, relying on theorems 6.1 and 6.2. $\square$

Chapter 5

Solving k-register synthesis problem

This chapter's main theorem is a solution to problem 3.6 from §3.2:

Theorem 5.1. synthesis of DRA_k

For every fixed $k \in \mathbb{N}$, the k-register synthesis problem is computable.

Central to its proof are two reductions extracted into lemmas below. First, we move from k-register synthesis problem to an intermediate ad-hoc *atom-blind synthesis problem*.

Lemma 5.2. eliminating winning controller's registers

For every $k \in \mathbb{N}$, the k-register synthesis problem reduces to the atom-blind synthesis problem.

This is done by introducing a protocol which, conceptually, allows Bob to outsource handling his registers, rendering them redundant in Bob's controllers. Subsequently, eliminating atoms altogether, we pose an equivalent synthesis problem over finite alphabets:

Lemma 5.3. eliminating atoms

The atom-blind synthesis problem reduces to the synthesis problem over finite alphabets.

Thus, we reduced to a problem which is amenable to solving using the Büchi-Landweber result (theorem 2.30).

Proof of theorem 5.1.

Trivial consequence of lemmas 5.2 and 5.3 and theorem 2.30. Based on a game provided in the input, using the above reductions, we first construct an intermediate instance of the atom-blind synthesis problem and then an instance of synthesis over finite alphabets. Theorem 2.30 allows us to solve the reduced problem. If a winning controller is synthesized, we go back in the chain of reductions, eventually constructing a winning k-register controller. □

We first define the atom-blind synthesis problem, while the above lemmas are proved in the last two sections of this chapter.

5.1 Atom-blind synthesis problem

In a nutshell, this problem asks for atom-blind controllers in a register synthesis game, at first glance severely limiting the playing ability of Bob.

Fix a register synthesis game $G = G^{\mathbb{A}}_{A,B}(W)$. An *atom-blind strategy* of Bob in G is any function of type $f : A^+ \to B$ (as opposed to the type $(A \times \mathbb{A})^+ \to B$ of the unrestricted notion of strategy). We say that a play

$$\pi = (a_1, b_1, \alpha_1)(a_2, b_2, \alpha_2)(a_3, b_3, \alpha_3) \cdots \in (A \times B \times \mathbb{A})^\omega$$

is f-conformant whenever $b_n = f(a_1 a_2 \dots a_n)$ for every $n \in \mathbb{N}_{\geq 1}$. Such strategies correspond to $\mathcal{S} \in \text{DLTS}(A, B)$ in a natural way (cf. §2.7.1). When $\mathcal{S}$ is finite, it belongs to $\text{DFA}(A, B)$ and is called an *atom-blind controller*. Its induced strategy—mapping every $w \in A^+$ to the last letter of $\mathcal{S}(w)$—is denoted as $\mathsf{Strategy}(\mathcal{S})$. An atom-blind controller $\mathcal{S}$ for Bob is winning in G, whenever no $\mathsf{Strategy}(\mathcal{S})$-conformant run belongs to the winning language W of Alice.

Problem 5.4. atom-blind synthesis problem

INPUT:	a register synthesis game $G^{\mathbb{A}}_{A,B}(W)$
OUTPUT:	▶ If exists, return a winning atom-blind controller $\mathcal{S} \in \text{DRA}(A, B)$ for Bob.
	▶ Otherwise, NIL.

5.2 Eliminating winning controller's registers

Reduction idea

Introduction of a carefully crafted *register outsourcing protocol* for Bob allows us to reduce the k-register synthesis problem to the atom-blind synthesis problem. Given a game G—an instance of k-register synthesis—we construct a modified game $\text{OUT}_k(G)$. Therein, Bob, instead of manipulating registers directly, delegates handling of them to Alice. In every turn, he tells Alice if and where to store the current atom. An updated winning condition obliges Alice to inform Bob about the orbit of the current register valuation $\nu \in \mathbb{A}_\perp^{\mathtt{X} \cup \{\mathtt{y}\}}$ (where $\mathtt{X}$ is the set of k registers, $\mathtt{y}$—the current atom). Should Alice provide incorrect information, it will prevent her from winning. This way, Bob retains all the information and control, enabling him to play exactly the same way as in the original game G, where he had registers. Consequently, winning atom-blind controllers exist in $\text{OUT}_k(G)$, whenever winning k-register controllers exist in G.

Proof structure

We begin by defining game $\text{OUT}_k(G)$ and its underlying register outsourcing protocol. Crucially to the correctness of the reduction, we observe that:

Lemma 5.5. the definition of $\text{OUT}_k(G)$ is correct

For every register synthesis game G, winning condition of $\text{OUT}_k(G)$ is an NRA^{g} language.

Therefore, $\text{OUT}_k(G)$ is also a register synthesis game. The reduction is concluded with:

Lemma 5.6. $\mathcal{S}$ is winning $\iff$ $F(\mathcal{S})$ is winning

For every register synthesis game G and $k \in \mathbb{N}$, there is a winning atom-blind controller $\mathcal{S}$ in $\textsf{Out}_k(G)$ if, and only if, there exists a winning k-register controller in G. Moreover, having the former, the latter can be effectively constructed.

Proof of lemma 5.2.

Fix $k \in \mathbb{N}$ and a register synthesis game G. Construct game $\textsf{Out}_k(G)$, which is a register synthesis game due to lemma 5.5; solve the corresponding at synthesis problem. Using lemma 5.6, if a winning controller $\mathcal{S}$ is found, compute the k-register controller for G, otherwise return Nil. □

Reduction

Definition 5.7. definition of $\textsf{Out}_k(G)$

Fix $k \in \mathbb{N}$ and a register synthesis game $G^{\mathbb{A}}_{A,B}(W)$. Let $\mathtt{X} = \{\mathtt{x}_1, \mathtt{x}_2, \ldots, \mathtt{x}_k\}$ be a k-element set of register names. $\textsf{Out}_k(G)$ is a register synthesis game $G^{\mathbb{A}}_{A',B'}(W')$, where:

$$A' := A \times \textsf{MaxConstr}(\mathtt{X} \cup \{\mathtt{y}\}), \tag{5.1}$$
$$B' := B \times \mathcal{P}(\mathtt{X}), \tag{5.2}$$
$$W' := \phi^{-1}(W) \cap R. \tag{5.3}$$

The winning condition W' has two components, which are defined below.

First one, $\phi^{-1}(W)$, simply adapts the original condition W to new alphabets, using a projection function $\phi : (A' \times B' \times \mathbb{A}) \to (A \times B \times \mathbb{A})$ which ignores the extra data included in the letters: $\phi((a, \_), (b, \_), \alpha) = (a, b, \alpha)$. ϕ is extended homomorphically on finite and infinite plays.

Second one, R, is a language of *correct register traces*. It accepts a play

$$\pi = ((\_, \varphi_1), (\_, \mathtt{Y}_1), \alpha_1)((\_, \varphi_2), (\_, \mathtt{Y}_2), \alpha_2) \cdots \in \textsf{Plays}(\textsf{Out}_k(G)) \tag{5.4}$$

whenever the constraints $\varphi_\bullet$ provided by Alice match register valuations inferred from atoms $\alpha_\bullet$ and reset sets $\mathtt{Y}_\bullet$. Formally, it verifies that in every round $i \in \mathbb{N}_{\geq 1}$

$$\mu_{i-1}[\mathtt{y} \mapsto \alpha_i] \vDash \varphi_i, \tag{5.5}$$

where valuations $\mu_\bullet \in \mathbb{A}_\perp^{\mathtt{X}}$ are defined in line with the semantics of a register automaton

$$\mu_0 := \lambda x.\perp \tag{5.6}$$
$$\mu_i := \mu_{i-1}[\mathtt{Y}_i \mapsto \alpha_i]. \tag{5.7}$$

To make sure that $\textsf{Out}_k(G)$ is defined properly, we need to prove lemma 5.5:

Proof of lemma 5.5.

We aim at showing that winning condition W' of $\textsf{Out}_k(G)$ is a $\textsf{NRA}^{\mathsf{g}}$ language. As the class of $\textsf{NRA}^{\mathsf{g}}$ is closed under inverse homomorphic images and intersections, it only remains to make sure that R is an $\mathsf{NRA}^{\mathsf{g}}$ language. As it turns out, it can even be recognised by a DRA. We construct $\mathcal{R} \in \mathsf{DRA}_k$ with two states $\textsc{Ok}, \textsc{Err}$, the former being accepting and latter—rejecting. The automaton updates its registers at Bob's request and stays in state $\textsc{Ok}$ while the constraints provided by Alice are true

$$(\textsc{Ok} \xRightarrow{((\_,\varphi),(\_,\mathtt{Y})),\varphi,\mathtt{Y}} \textsc{Ok}).$$

If Alice plays a maximal constraint φ different than the actual φ', $\mathcal{R}$ changes state to $\textsc{Err}$ and stays there

$$(\textsc{Ok} \xRightarrow{((\_,\varphi),(\_,\mathtt{Y})),\varphi',\mathtt{Y}} \textsc{Err})$$

$$(\textsc{Err} \Rightarrow \textsc{Err}).$$

It is easy to see that the above automaton recognises R. □

Proof of lemma 5.6.

Fix $k \in \mathbb{N}$, a game $G = G^{\mathbb{A}}_{A,B}(W)$, and a k-element set of register names $\mathtt{X}$. Let $G' = G^{\mathbb{A}}_{A',B'}(W') = \textsc{Out}_k(G)$.

We first define a computable bijection F, which maps Bob's atom-blind controllers in G' to controllers in G. Then, we complete the proof by showing that $F(\mathcal{S})$ is winning if, and only if, $\mathcal{S}$ is winning.

Definition of bijection F

Let $F : \mathsf{DFA}(A', B') \hookrightarrow\!\!\!\!\rightarrow \mathsf{DRA}^{\mathsf{max}}_k(A, B)$ be defined as follows

$$F(A', B', L, l_{\mathsf{ini}}, \delta) = \Big(\mathtt{X}, A, B, L, l_{\mathsf{ini}}, \Big\{(p \xRightarrow{a/b,\varphi,\mathtt{Y}} q) \;\Big|\; (p \xRightarrow{(a,\varphi)/(b,\mathtt{Y})} q) \in \delta\Big\}\Big)$$

F is clearly computable and is a bijection.

$\mathcal{S}$ wins in G' $\implies$ $F(\mathcal{S})$ wins in G

Fix a winning atom-blind controller $\mathcal{S} = (A', B', L, l_{\mathsf{ini}}, \delta) \in \mathsf{DFA}(A', B')$. We will show that $F(\mathcal{S})$ is winning in G. Let π be any $F(\mathcal{S})$-conformant play of G

$$\pi \;=\; (a_1, b_1, \alpha_1) \qquad (a_2, b_2, \alpha_2) \qquad (a_3, b_3, \alpha_3) \qquad \ldots$$

We have to prove that $\pi \notin W$. Let σ be a run of $F(\mathcal{S})$ corresponding to π, and ρ—its designating sequence of transition rules

$$\sigma \;=\; (l_0, \mu_0) \xrightarrow{(a_1,\alpha_1)/b_1,} (l_1, \mu_1) \xrightarrow{(a_2,\alpha_2)/b_2,} (l_2, \mu_2) \xrightarrow{(a_3,\alpha_3)/b_3,} \ldots$$

$$\rho \;=\; l_0 \xRightarrow{a_1/b_1,\varphi_1,\mathtt{Y}_1} l_1 \xRightarrow{a_2/b_2,\varphi_2,\mathtt{Y}_2} l_2 \xRightarrow{a_3/b_3,\varphi_3,\mathtt{Y}_3} \ldots$$

Using definition of F we conclude that σ' below is a run of $\mathcal{S}$, corresponding to a play π':

$$
\begin{aligned}
\sigma' = \quad & l_0 \xrightarrow{(a_1,\varphi_1)/(b_1,\mathtt{Y}_1)} l_1 \xrightarrow{(a_2,\varphi_2)/(b_2,\mathtt{Y}_2)} l_2 \xrightarrow{(a_3,\varphi_3)/(b_3,\mathtt{Y}_3)} \dots \\
\pi' = \quad & \big((a_1,\varphi_1),(b_1,\mathtt{Y}_1),\alpha_1\big)\ \big((a_2,\varphi_2),(b_2,\mathtt{Y}_2),\alpha_2\big)\big((a_3,\varphi_3),(b_3,\mathtt{Y}_3),\alpha_3\big)\ \dots
\end{aligned}
$$

Since $\mathcal{S}$ is winning, the $\mathcal{S}$-conformant play π' does not belong to W', thus $\pi \notin W$, as required.

$F(\mathcal{S})$ wins in $G \Rightarrow \mathcal{S}$ wins in G'

Here, we fix a winning k-register controller $\mathcal{S} = (\mathtt{X}, A, B, L, l_{\mathsf{ini}}, \delta) \in \mathsf{DRA}_k(A, B)$, and need to show that $F^{-1}(\mathcal{S})$ is winning in G'. W.l.o.g., due to lemma 2.21, we may assume that $\mathcal{S} \in \mathsf{DRA}_k^{\mathsf{max}}(A, B)$. We reuse the notations $\pi', \sigma', \pi, \sigma, \rho$ from the first part, but we will quantify them differently.

Let π' be an arbitrary $F^{-1}(\mathcal{S})$-conformant play of G'. We have to prove that $\pi' \notin W'$. It is trivially the case when $\pi' \notin R$, so let us assume otherwise. Let σ' be the run of $F^{-1}(\mathcal{S})$ associated to π'. Consider a sequence σ of transitions of $[\![\mathcal{S}]\!]$, as defined above. Observe that since $\pi' \in R$, we know that $\mu_{i-1}[\mathtt{y} \mapsto \alpha_i] \vDash \varphi_i$ for all i. Therefore, σ is a correct run of $\mathcal{S}$, with $\mathsf{runToPlay}(\sigma) = \pi$. The $\mathcal{S}$-conformant play π is winning, therefore $\pi \notin W$ and, consequently, $\pi' \notin \phi^{-1}(W)$, which finishes the proof. □

5.3 Eliminating atoms

Reduction idea

The main tool in this reduction is the following lemma:

Lemma 5.8. folklore knowledge

For every $\mathcal{S} \in \mathsf{NRA}^{\mathsf{g}}(\Sigma)$, one can construct $\mathcal{S}' \in \mathsf{NFA}(\Sigma)$, such that

$$\mathcal{L}(\mathcal{S}') = \{\mathsf{proj}_{\mathbb{A}}(w) \mid w \in \mathcal{L}(\mathcal{S})\}$$

Analogous property applies also to ω-languages $\mathcal{L}^\omega(\mathcal{S}), \mathcal{L}^\omega(\mathcal{S}')$.

Proof of lemma 5.8 (sketch).

The proof is analogous to a timed automata result by Alur and Dill [2], in which they construct NFA recognising $\mathsf{proj}_{\mathbb{T}}(\mathcal{S})$ for $\mathcal{S} \in \mathsf{NTA}$. Here, given $\mathcal{S} \in \mathsf{NRA}^{\mathsf{g}}$, one constructs an 'orbit automaton' $\mathcal{S}'$—NFA with control locations $L \times \textsc{MaxConstr}(\mathtt{X})$, L being the original set of locations of $\mathcal{S}$. For each transition

$$(\ell_1, \mu_1) \xrightarrow{\sigma,\alpha} (\ell_2, \mu_2) \tag{5.8}$$

of $[\![\mathcal{S}]\!]$, $\mathcal{S}'$ has a transition

$$(\ell_1, \varphi_1) \xrightarrow{\sigma} (\ell_2, \varphi_2), \tag{5.9}$$

where φ_1, φ_2 are unique maximal constraints such that $\mu_1 \vDash \varphi_1$, $\mu_2 \vDash \varphi_2$. Crucially, for every transition as in eq. (5.9), and $\nu_1 \in \mathbb{A}_\perp^{\mathtt{X}}$, $[\![\mathcal{S}]\!]$ has some transition

$$(\ell_1, \psi(\mu_1)) \xrightarrow{\sigma, \psi(\alpha)} (\ell_2, \psi(\mu_2))$$

obtained from eq. (5.8) by applying an atom automorphism $\psi \in \mathsf{Aut}(\mathbb{A})$ which maps μ_1 to ν_1. The statement of the lemma follows easily from this fact. □

Reduction

Proof of lemma 5.3.

Fix a register synthesis game $G = G^{\mathbb{A}}_{A,B}(W)$. Let $G' = G_{A,B}(\mathsf{proj}_{\mathbb{A}}(W))$, where the automaton for $\mathsf{proj}_{\mathbb{A}}(W)$ follows from lemma 5.8. Observe that atom-blind controllers in G and controllers in G' both have type $\mathsf{DFA}(A, B)$. Fix $\mathcal{S} \in \mathsf{DFA}(A, B)$. We will show that $\mathcal{S}$ is winning in G if, and only if, it is winning in G'.

The '⇒' implication

Assume $\mathcal{S}$ is winning in G. Take an arbitrary $\mathcal{S}$-conformant play π' of G'. Assume that $\pi' \in \mathsf{proj}_{\mathbb{A}}(W)$. Therefore, there exists a play $\pi \in (A \times B \times \mathbb{A})^*$ such that $\pi \in W$. Since, $\mathcal{S}$ is atom-blind, π is $\mathcal{S}$-conformant and losing for Bob, contradiction.

The '⇐' implication

Assume $\mathcal{S}$ is winning in G'. Take an arbitrary $\mathcal{S}$-conformant play $\pi \in (A \times B \times \mathbb{A})^*$ of G. Assume that is losing for Bob in G, i.e., $\pi \in W$. But then $\mathsf{proj}_{\mathbb{A}}(\pi') \in \mathsf{proj}_{\mathbb{A}}(W)$, which is $\mathcal{S}$-conformant and losing for Bob in G', contradiction. □

Chapter 6

Solving k-clock timed synthesis problems

This chapter tackles k-clock timed synthesis problems:

- problem 3.10: k-clock m-constrained timed synthesis problem, and
- problem 3.9: k-clock timed synthesis problem—where the maximal constant m used in clock constraints is not specified in advance.

We prove computability of both:

Theorem 6.1. synthesis of $\text{DTA}_{k,m}$

For every fixed $k, m \in \mathbb{N}$, the k-clock m-constrained timed synthesis problem is computable.

Theorem 6.2. synthesis of DTA_k

For every fixed $k \in \mathbb{N}$, the k-clock timed synthesis problem is computable.

Structure

Above theorems are achieved in four steps, corresponding to §§6.1, 6.2.1, 6.2.2 and 6.3.

- *§6.1 Simplifying assumptions*
  Firstly, we show that certain simplifying assumptions about the winning condition W of any timed synthesis game can be made; i.e., that all words in W
  - are strictly monotonic, and
  - *zero-starting*: $\tau_1 = 0$ for any $(a_1, \tau_1)(a_2, \tau_2) \cdots \in W$.
- *§6.2 Elliminating winning controller's clocks*
  - *§6.2.1 Case of bounded constants in clock constraints*
    Then, we present the main technical construction in the proof of theorem 6.1, showing that k-clock m-constrained timed synthesis reduces to the degenerate 0-clock timed synthesis problem.

  - ▶ *§6.2.2 Case of unbounded constants in clock constraints*
    As a relatively easy extension of proof of theorem 6.1, we obtain theorem 6.2, reducing once again to the 0-clock timed synthesis problem.

- ▶ *§6.3 Elliminating time*
  Finally, we prove that degenerate 0-clock timed synthesis problem reduces to synthesis over finite alphabets, computable due to Büchi-Landweber theorem theorem 2.30.

The computability results of this section are tight, since timed synthesis—even in its decision variant—is undecidable when the number of clocks is not fixed (c.f. theorem 8.2).

Similarities to solving the k-register synthesis problem

The general structure of the proof bears some resemblance to the register case. Here, likewise, first reduction removes the clocks from Bob's winning controllers (cf. §5.2 Eliminating winning controller's registers). Then, both proofs conclude with construction of games with ω-regular winning condition.

Nevertheless, the ways of outsourcing controller's registers and clocks to Alice are fundamentally different. This is because the adapted reasoning for registers would only apply to the k-clock m-constrained timed synthesis problem. To observe this, recall that in the intermediate atom-blind game, Alice plays maximal constraints corresponding to orbits of register valuations. A natural timed counterpart of that are k, m-regions, which—if used in the 0-clock synthesis—would result in sewing a fixed m into the very alphabet of Alice. Lifting the 'm-constrained' restriction required a more refined reduction. In it, instead of a single k, m-region, Bob, after every simulated reset of one of the k clocks, receives from Alice a series of 'ticks' at intervals of one time unit. In m-constrained synthesis the limit on the length of such series is m, whereas in unconstrained variant we only require that series to be finite.

6.1 Simplifying assumptions

6.1.1 Zero-starting winning conditions

A timed language $W \subseteq \mathbb{T}^\omega(\Sigma)$ is *zero-starting* if, and only if, all its nonempty words $(\_, \tau_0)(\_, \tau_1)\cdots \in W$ satisfy $\tau_0 = 0$.

Lemma 6.3. the zero-starting assumption is safe

For any timed synthesis game G there exists a game G' with zero-starting condition, such that Bob has a winning controller $\mathcal{M} \in \mathsf{DTA}_{k,m}$ for G if, and only if, he has a winning controller $\mathcal{M}' \in \mathsf{DTA}_{k,m}$ in G'.

Proof of lemma 6.3.

Let $G = G^{\mathbb{T}}_{A,B}(W)$ be a timed synthesis game, where $W \subseteq \mathbb{T}^\omega(A \times B)$. We design an equivalent timed synthesis game $G' = G^{\mathbb{T}}_{A',B}(W')$, where actions of Alice are in $A' = A \cup \{\triangleright\}$, and the zero-starting winning condition is $W' = \{(\triangleright, b, 0)\cdot w | b \in B, w \in W\}$.

$\mathcal{M}'$ is obtained from $\mathcal{M}$ by responding arbitrarily to every $\triangleright$, and conversely, $\mathcal{M}$ is obtained from $\mathcal{M}' = (\mathtt{X}, A, B, L', \ell'_0, \delta')$ by restricting to alphabet A and letting the initial location be the unique ℓ_0 such that $\delta'(\ell'_0, \triangleright, \mathtt{r}_0) = (\ell_0, \_, \_)$. □

6.1.2 Strictly monotonic winning conditions

Lemma 6.4. *the assumption of strict monotonicity is safe*

Solving a timed game $G = G^{\mathbb{T}}_{A,B}(W)$ with a monotonic winning condition $W \subseteq \mathbb{T}^\omega(A \times B)$ reduces to solving one $G' = G^{\mathbb{T}}_{A',B}(W')$ with a strictly monotonic winning condition $W' \subseteq \mathbb{T}^\omega(A' \times B)$.

Proof of lemma 6.4.

We take Alice's action to be in $A' := A \times \{0, 1\}$. Consider the function ϕ mapping a play in G' of the form

$$\pi' = ((a_0, f_0), b_0, \tau'_0)((a_1, f_1), b_1, \tau'_1) \cdots \in \mathbb{T}^\omega(A' \times B) \tag{6.1}$$

to a corresponding play in G

$$\pi = \phi(\pi') = (a_0, b_0, \tau_0)(a_1, b_1, \tau_1) \cdots \in \mathbb{T}^\omega(A \times B) \tag{6.2}$$

where the new sequence of timestamps $\tau_0\tau_1 \cdots \in \mathbb{Q}^\omega_{\geq 0}$ is defined as $\tau_0 = \tau'_0$ and, inductively, $\tau_{i+1} = \tau_i$ if $f_{i+1} = 0$, and $\tau_{i+1} = \tau'_{i+1}$ otherwise. Let $W_< = \{\pi' \mid \tau'_0 < \tau'_1 < \cdots\}$ be the language of strictly monotonic plays. The winning condition in G' is then

$$W' = \phi^{-1}(W) \cap W_<.$$

Claim 6.5.

Bob has a winning controller $\mathcal{M} \in \mathsf{DTA}_{k,m}(A, B)$ in G if, and only if, he has a controller $\mathcal{M}' \in \mathsf{DTA}_{k,m}(A', B)$ in G'.

The '⇒' implication

Let $\mathcal{M} = (\mathtt{X}, A, B, L, \ell_0, \delta) \in \mathsf{DTA}_{k,m}(A, B)$ be a winning controller for Bob in G. We build a winning controller $\mathcal{M}' = (\mathtt{X}, A', B, L', \ell'_0, \delta')$ for Bob in G' as follows. Control locations are $L' = L \times \mathsf{Reg}(k, m)$, the initial location is $\ell'_0 = (\ell_0, \mathtt{r}_0)$, and the transition relation δ' is defined, for every input $(\ell, \varphi), (a, f), \varphi'$, as

$$\delta'((\ell, \varphi), (a, f), \varphi') = \begin{cases} ((\ell', \varphi), b, \mathtt{Y}) & \text{if } f = 0 \text{ and } \delta(\ell, a, \varphi) = (\ell', b, \mathtt{Y}), \\ ((\ell', \varphi'), b, \mathtt{Y}) & \text{if } f = 1 \text{ and } \delta(\ell, a, \varphi') = (\ell', b, \mathtt{Y}). \end{cases}$$

Assume π' is an $\mathcal{M}'$-conformant play as in eq. (6.1). If it is not strictly monotonic, then $\pi' \notin W'$ and we are done. Otherwise, assume π' is strictly monotonic. Towards reaching a contradiction, assume $\pi' \in \phi^{-1}(W)$. Therefore, $\pi = \phi(\pi') \in W$ as in eq. (6.2). By the definition of δ', π is $\mathcal{M}$-conformant, contradicting that $\mathcal{M}$ is winning.

The '$\Leftarrow$' implication

Let $\mathcal{M}' = (\mathtt{X}, A', B, L', \ell'_0, \delta') \in \mathsf{DTA}_{k,m}(A', B)$ be a winning controller for Bob in G'. We assume without loss of generality that Bob remembers the input region when the flag $f = 1$ was played last. Thus, locations in L' are of the form (ℓ, φ). We build a winning controller $\mathcal{M} = (\mathtt{X}, A, B, L', \ell'_0, \delta)$ for Bob in G where

$$\delta((\ell, \varphi), a, \varphi') = \begin{cases} ((\ell', \varphi), b, \mathtt{Y}) & \text{if } \varphi' = \varphi \text{ and } \delta'((\ell, \varphi), (a, 0), \varphi') = ((\ell', \varphi), b, \mathtt{Y}), \\ ((\ell', \varphi'), b, \mathtt{Y}) & \text{if } \varphi' \neq \varphi \text{ and } \delta'((\ell, \varphi), (a, 1), \varphi') = ((\ell', \varphi'), b, \mathtt{Y}). \end{cases}$$

Let π be a $\mathcal{M}$-conformant play and assume towards a contradiction that $\pi \in W$. We can choose sufficiently small increments in order to make all sequences of equal timestamps in π become strictly monotonic, and choose the flags f_i accordingly, and obtain a play π' such that $\pi = \phi(\pi')$. By the definition of δ, π' is $\mathcal{M}'$-conformant. But $\pi' \in W'$, which contradicts with the assumption that $\mathcal{M}'$ is winning in G'. □

6.2 Elliminating winning controller's clocks

6.2.1 Case of bounded constants in clock constraints

In this section we prove theorem 6.1 by reducing the k-clock m-constrained timed synthesis problem to 0-clock timed synthesis problem, which is computable by lemma 6.13. This is the most technically involved section. Modifying slightly the reduction presented here, we will show computability of the k-clock timed synthesis problem in §6.2.2.

Let $\mathtt{X}$ be a fixed set of clocks of size $|\mathtt{X}| = k$ and let $m \in \mathbb{N}$ be a fixed bound on constants.

Intuition

We reduce the k-clock m-constrained synthesis problem to the 0-clock synthesis problem by designing a protocol in which Bob, to compensate his inability to measure time elapse, can *request* certain clocks to be *tracked*. Intuitively—omitting some corner cases—Alice's winning condition obliges her to play so called 'improper actions' of the form $(\square, \_)$ exactly one time unit after a corresponding request. As a reaction to improper action of Alice, Bob may renew an earlier request; otherwise that request *expires*. In case of m-constrained synthesis, the number of consecutive tracking request renewals is bounded by $m-1$, allowing Bob to measure time up to m. In the following §6.2.2, this requirement is replaced with weaker one—that there is no infinite sequence of renewals. It turns out that this solves the unconstrained k-clock timed synthesis problem (3.9).

Fractional regions

Let $\mathsf{fract}(\mathtt{x})$ stand for the fractional part of the value of a clock $\mathtt{x}$. For $\mathtt{Y}_1, \mathtt{Y}_2 \subseteq \mathtt{X}$, two (partial) clock valuations $\mu \in \mathbb{Q}_{\geq 0}^{\mathtt{Y}_1}, \nu \in \mathbb{Q}_{\geq 0}^{\mathtt{Y}_2}$ are *fractional region equivalent* if $\mathtt{Y}_1 = \mathtt{Y}_2$ and they exhibit the same relations between fractional parts of clocks, i.e., for all $\mathtt{x}, \mathtt{x}' \in \mathtt{Y}_1$ the following holds:

- $\mu \models \mathsf{fract}(\mathtt{x}) < \mathsf{fract}(\mathtt{x}')$ if, and only if, $\nu \models \mathsf{fract}(\mathtt{x}) < \mathsf{fract}(\mathtt{x}')$, and
- $\mu \models \mathsf{fract}(\mathtt{x}) = 0$ if, and only if, $\nu \models \mathsf{fract}(\mathtt{x}) = 0$.

By a *(partial) fractional* $\mathtt{X}$*-region* $\mathtt{f}$ we mean an equivalence class of this equivalence relation. All elements $\mu \in \mathbb{Q}^{\mathtt{Y}}_{\geq 0}$ in $\mathtt{f}$ have the same domain $\mathtt{Y}$, which we denote by $\mathsf{dom}(\mathtt{f}) = \mathtt{Y}$. Let $\mathsf{integer}(\mathtt{f}) = \{\mathtt{x} \in \mathsf{dom}(\mathtt{f}) \mid \mathtt{f} \models \mathsf{fract}(\mathtt{x}) = 0\}$, i.e., clocks of $\mathsf{dom}(\mathtt{f})$ with integer value. Let $\mathsf{FReg}(\mathtt{X})$ be the set of all fractional $\mathtt{X}$-regions, including the empty one $\mathtt{f}_0$ with $\mathsf{dom}(\mathtt{f}_0) = \varnothing$. For $\mathtt{r} \in \mathsf{Reg}(\mathtt{X}, m)$ and $\mathtt{f} \in \mathsf{FReg}(\mathtt{X})$, we say that $\mathtt{f}$ *agrees* with $\mathtt{r}$ if they give the same answer for clocks $\mathtt{x}, \mathtt{y} \in \mathsf{dom}(\mathtt{f})$:

- $\mathtt{f} \models \mathsf{fract}(\mathtt{x}) < \mathsf{fract}(\mathtt{y})$ if, and only if, $\mathtt{r} \models \mathsf{fract}(\mathtt{x}) < \mathsf{fract}(\mathtt{y}) \vee \mathtt{x} > m \vee \mathtt{y} > m$;
- $\mathtt{f} \models \mathsf{fract}(\mathtt{x}) = 0$ if, and only if, $\mathtt{r} \models \mathsf{fract}(\mathtt{x}) = 0 \vee \mathtt{x} > m$.

The successor relation between regions induces a corresponding relation between fractional regions: $\mathtt{f} \preceq \mathtt{f}'$ whenever $\mathsf{dom}(\mathtt{f}) = \mathsf{dom}(\mathtt{f}')$, $\mathtt{f}$ agrees with some $\mathtt{r}$, $\mathtt{f}'$ agrees with some $\mathtt{r}'$, and $\mathtt{r} \preceq \mathtt{r}'$. The immediate successor is the minimal $\mathtt{f}'$ with $\mathtt{f} \prec \mathtt{f}'$. Finally, the *successor region of* $\mathtt{r}$ *agreeing with* $\mathtt{f}$ is $\mathsf{succ}_{\mathtt{X},m}(\mathtt{r}, \mathtt{f}) = \min_{\preceq}\{\mathtt{r}' \succeq \mathtt{r} \mid \mathtt{f} \text{ agrees with } \mathtt{r}'\}$. In the sequel we apply clock resets also to regions $\mathtt{r}[\mathtt{Y} \mapsto 0]$ and fractional regions.

The reduction

Let the original game $G = G^{\mathbb{T}}_{A,B}(W)$ have action alphabets A, B and Alice's winning condition $W \subseteq \mathbb{T}^{\omega}(A \times B)$. Thanks to §§6.1.1 and 6.1.2 we assume w.l.o.g. that W is both strictly monotonic and zero starting. We design a new game $G' = G^{\mathbb{T}}_{A',B'}(W'_{k,m})$ as follows. We take as the new action alphabets the sets

$$A' \;=\; (A \cup \{\square\}) \times \mathsf{FReg}(\mathtt{X}) \quad \text{and} \quad B' \;=\; (B \cup \{\square\}) \times \mathcal{P}(\mathtt{X}). \tag{6.3}$$

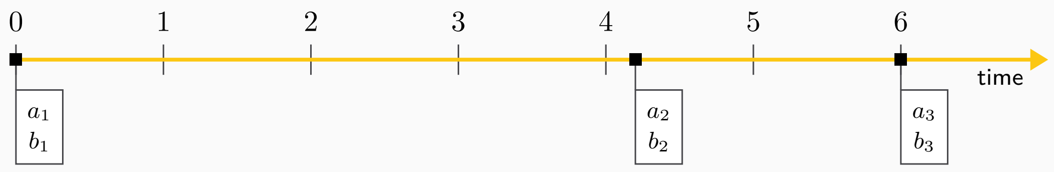
The players' action sets A', B' depend only on the set of clocks $\mathtt{X}$ and do not depend on the maximal constant m. Moves of the form $(\square, \_)$ are *improper* and the other ones (i.e., those involving A or B components) are *proper*. Let an infinite play be of the form

$$\pi = (a'_1, b'_1, \tau_1)(a'_2, b'_2, \tau_2) \cdots \in \mathbb{T}^{\omega}(A' \times B'), \tag{6.4}$$

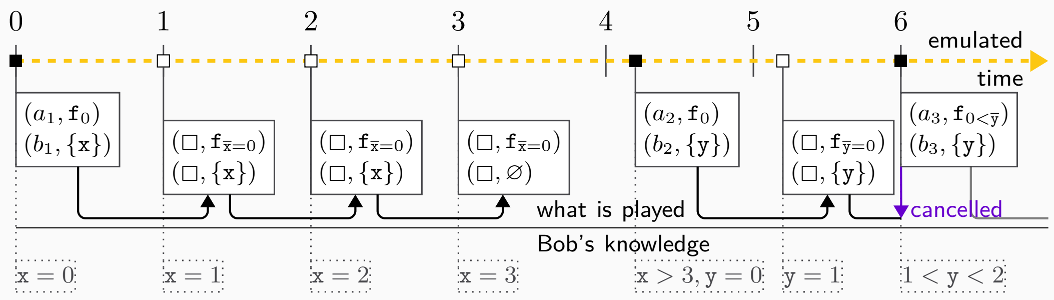
where $a'_i = (a_i, \mathtt{f}_i)$ and $b'_i = (b_i, \mathtt{Y}_i)$. The domain $\mathtt{T}_i = \mathsf{dom}(\mathtt{f}_i)$ of a fractional region denotes the clocks *tracked* at time τ_i, i.e., those for which Alice needs to provide expiry information. Sets $\mathtt{Y}_\bullet$ denote clocks which Bob wants to be still tracked: by an $\mathtt{x}$*-request at time* τ_i we mean a Bob's move b'_i with $\mathtt{x} \in \mathtt{Y}_i$. An $\mathtt{x}$-request at time τ_i is *cancelled* if there is another $\mathtt{x}$-request for the same clock at some time $\xi \in (\tau_i, \tau_i + 1)$. An *improper improper-request chain* starting at time τ_i of length $l \geq 1$ is a sequence of improper non-cancelled $\mathtt{x}$-requests at times τ_i, $\tau_i + 1$, ..., $\tau_i + l - 2$, followed by an improper (but possibly cancelled) $\mathtt{x}$-request at time $\tau_i + l - 1$. Likewise one defines an infinite improper $\mathtt{x}$-request chain starting at time τ_i.

Example 6.6. replacing clocks with chains of tracking requests

Before defining the winning set $W'_{k,m}$ formally, we illustrate the underlying idea. Consider the following partial play $(a_1, b_1, 0)\,(a_2, b_2, 4.2)\,(a_3, b_3, 6) \in \mathbb{T}(A \times B)$ in G:



In G', Bob demands Alice to provide clock expiry information. Let $\mathtt{X} = \{\mathtt{x}, \mathtt{y}\}$ and $m = 3$. Suppose Bob wants to make sure that a_2 comes at time > 3. To this end, he makes an $\mathtt{x}$-request chain of length 3 (we write $\bar{\mathtt{x}}$ instead of $\mathsf{fract}(\mathtt{x})$; $\mathtt{f}_\phi$ denotes the fractional $\mathtt{X}$-region agreeing with ϕ):


0
1
2
3
4
5
6
emulated time
$(a_1, \mathtt{f}_0)$ $(b_1, \{\mathtt{x}\})$
$(\square, \mathtt{f}_{\bar{\mathtt{x}}=0})$ $(\square, \{\mathtt{x}\})$
$(\square, \mathtt{f}_{\bar{\mathtt{x}}=0})$ $(\square, \{\mathtt{x}\})$
$(\square, \mathtt{f}_{\bar{\mathtt{x}}=0})$ $(\square, \varnothing)$
$(a_2, \mathtt{f}_0)$ $(b_2, \{\mathtt{y}\})$
$(\square, \mathtt{f}_{\bar{\mathtt{y}}=0})$ $(\square, \{\mathtt{y}\})$
$(a_3, \mathtt{f}_{0<\bar{\mathtt{y}}})$ $(b_3, \{\mathtt{y}\})$
what is played
cancelled
Bob's knowledge
$\mathtt{x} = 0$
$\mathtt{x} = 1$
$\mathtt{x} = 2$
$\mathtt{x} = 3$
$\mathtt{x} > 3, \mathtt{y} = 0$
$\mathtt{y} = 1$
$1 < \mathtt{y} < 2$

The length of an $\mathtt{x}$-chain at any given moment corresponds to the integral part of $\mathtt{x}$; the expiry information for $\mathtt{x}$ is provided by Alice precisely when the fractional part of $\mathtt{x}$ is 0.

In order to define $W'_{k,m}$ it will be convenient to have the following additional data extracted from π. Let $\nu_0 = \lambda \mathtt{x}.0$ be the initial clock reset assignment, and, for $i \geq 0$, let

$$\nu_{i+1} = (\nu_i)[\mathtt{Y}_{i+1} \mapsto \tau_{i+1}]. \tag{6.5}$$

In words, every $\mathtt{x}$-request is interpreted as reset of clock $\mathtt{x}$. The winning condition $W'_{k,m}$ in the new game will impose, in addition to W, the following further conditions to be satisfied by Alice in order to win. Let $W^{\mathsf{A}}_k \subseteq \mathbb{T}^\omega(A' \times B')$ be the set of plays π as in (6.4) which are zero-starting ($\tau_1 = 0$), strictly monotonic and, for every $i \geq 1$:

1. For every $\mathtt{x} \in \mathtt{X}$, $\mathtt{x}$ is expired at time τ_i if, and only if, $\tau_i \geq 1$ and there is a non-cancelled $\mathtt{x}$-request at an earlier time $\tau_j = \tau_i - 1$.
2. Tracked clocks are consistent with requests: for every clock $\mathtt{x} \in \mathtt{X}$, $\mathtt{x}$ is tracked $\mathtt{x} \in \mathtt{T}_i$ at time τ_i if, and only if, there is an $\mathtt{x}$-request at an earlier τ_j with $\tau_i - 1 \leq \tau_j < \tau_i$.
3. The fractional regions are correct: $\mathtt{f}_i$ agrees with $[(\tau_i - \nu_{i-1})]_{\mathtt{X},m}$.

Thus the conditions above assure that Alice provides exactly all expiry information requested by Bob in a timely manner, and the fractional regions $\mathtt{f}_i$ are consistent with the requests and time elapse. Note that any play in W^{A}_k satisfies $0 < (\tau_i - \nu_i)(\mathtt{x}) \leq 1$ for every $i \geq 1$ and $\mathtt{x} \in \mathtt{T}_i$. Indeed, positivity is due to strict monotonicity, and the upper bound due to the conditions 1–3. Provided Alice satisfies W^{A}_k, she wins whenever Bob violates any of the conditions below: Let $W^{\mathsf{B}}_{k,m} \subseteq \mathbb{T}^\omega(A' \times B')$ be the set of plays π as in (6.4) such that

4. Bob plays a proper move if, and only if, Alice does so.

5. Every improper Bob's $\mathtt{x}$-request b'_i is a response to Alice's expiry information for $\mathtt{x}$: $\mathtt{Y}_i \subseteq \mathsf{integer}(\mathtt{f}_i)$. (Proper $\mathtt{x}$-requests are allowed unconditionally.)

6. For every clock $\mathtt{x} \in \mathtt{X}$, the length Bob's improper $\mathtt{x}$-request chains is $< m$. This is the only component in the winning condition which depends on m.

Consider the projection function $\phi : (A' \times B' \times \mathbb{Q}_{\geq 0}) \to (A \times B \times \mathbb{Q}_{\geq 0}) \cup \{\varepsilon\}$ such that

$$\phi((a, \_), (b, \_), \tau) = \begin{cases} \varepsilon & \text{if } a = \square \text{ or } b = \square \\ (a, b, \tau) & \text{if } a \in A \text{ and } b \in B \end{cases}$$

which is extended homomorphically on finite and infinite plays. The winning condition for Alice in G' is

$$W'_{k,m} \;=\; W_k^{\mathsf{A}} \cap \Big(\phi^{-1}(W) \cup \mathbb{T}^\omega(A' \times B') \setminus W_{k,m}^{\mathsf{B}}\Big). \tag{6.6}$$

Since W, W_k^{A}, are NTA^ε languages, and W_k^{A} and $W_{k,m}^{\mathsf{B}}$ are DTA_k languages over $A' \times B'$, thanks to the closure properties DTA and NTA^ε languages the winning condition $W'_{k,m}$ is an NTA^ε language.

Correctness of the reduction

In what follows, an *untimed controller* is a synonym for DTA_0. Then next two lemmas state the correctness of the reduction. Our assumption on strict monotonicity facilitates the correctness proof since we need not deal with simultaneous events.

Lemma 6.7.

If there is a winning controller $\mathcal{M} \in \mathsf{DTA}_{k,m}$ for Bob in G, then there is a winning untimed controller $\mathcal{M}' \in \mathsf{DTA}_0$ for Bob in G'.

Lemma 6.8.

If there is a winning untimed controller $\mathcal{M}' \in \mathsf{DTA}_0$ for Bob in G', then one can construct a winning controller $\mathcal{M} \in \mathsf{DTA}_{k,m}$ for Bob in G.

Proof of lemma 6.7.

Let $\mathcal{M} = (\mathtt{X}, A, B, L, \ell_0, \delta)$ be a winning k-clock m-constrained controller $\mathcal{M}$ for G with clocks $\mathtt{X} = \{\mathtt{x}_1, \ldots, \mathtt{x}_k\}$ and update function $\delta : L \times A \times \mathsf{Reg}(\mathtt{X}, m) \to L \times B \times \mathcal{P}(\mathtt{X})$. We define a winning untimed controller $\mathcal{M}' = (A', B', L', \textsc{Ini}, \delta')$ for G' with memory locations $L' = \{\textsc{Ini}\} \;\cup\; L \times \mathsf{Reg}(\mathtt{X}, m)$, where $\textsc{Ini}$ is the initial memory location, and remaining memory locations are of the form $(\ell, \mathtt{r})$, where $\ell \in L$ is the current memory location of $\mathcal{M}$ and $\mathtt{r} \in \mathsf{Reg}(\mathtt{X}, m)$ is the current region of $\mathcal{M}$'s clocks.

The update function $\delta' : L' \times A' \to L' \times B'$ (we omit regions and clock resets because $\mathcal{M}'$ has no clocks) is defined as follows. As long as the play is in W_k^{A}, we can assume that Alice starts with $((a, \mathtt{f}_0), \tau)$ and $\tau = 0$, due to the zero-starting restriction, which allows Bob to submit requests at time 0. Consequently, let $\delta'(\textsc{Ini}, (a, \mathtt{f})) = ((\ell', \mathtt{r}_0), (b, \mathtt{X}))$, where the next location ℓ' and the response b are determined by $\delta(\ell_0, a, \mathtt{r}_0) = (\ell', b)$, and the set $\mathtt{X}$ denotes a request to track all clocks. Then, for every $\ell, \mathtt{r}, a, \mathtt{f}$, let

$$\delta'((\ell, \mathtt{r}), (a, \mathtt{f})) = ((\ell', \mathtt{r}'), (b, \mathtt{Y})), \tag{6.7}$$

where the r.h.s. is defined as follows. Let $\mathtt{T} = \mathsf{dom}(\mathtt{f})$ be the currently tracked clocks, and $\mathtt{T}_0 = \mathsf{integer}(\mathtt{f}) \subseteq \mathtt{T}$ the currently expired ones. If $\mathtt{f}$ agrees with no successor region of $\mathtt{r}$ then Bob wins immediately because Alice is violating condition 3. Therefore, assume such a successor region $\hat{\mathtt{r}} = \mathsf{succ}_{\mathtt{X},m}(\mathtt{r}, \mathtt{f})$ exists. We do a case analysis based on whether Alice plays a proper or an improper move.

- Case $a \in A$ (proper move): Let $\delta(\ell, a, \hat{\mathtt{r}}) = (\ell', b, \mathtt{Y})$ thus defining ℓ' and $(b, \mathtt{Y})$ in (6.7). Take as the new region $\mathtt{r}' = \hat{\mathtt{r}}[\mathtt{Y} \mapsto 0]$.
- Case $a = \square$ (improper move): Let the response be also improper $b = \square$, the control location does not change $\ell' = \ell$, the new clocks to be tracked are the expired clocks with a short improper chain $\mathtt{Y} = \{\mathtt{x} \in \mathtt{T}_0 \mid \hat{\mathtt{r}} \models \mathtt{x} = 1 \vee \cdots \vee \mathtt{x} = m - 1\}$, and $\mathtt{r}' = \hat{\mathtt{r}}$.

Consider an infinite $\mathcal{M}'$-conformant run in G' (omitting clock valuations since $\mathcal{M}'$ has no clocks)

$$\rho' = \mathsf{INI} \xrightarrow{a_1', b_1', \tau_1} (\ell_1, \mathtt{r}_1) \xrightarrow{a_2', b_2', \tau_2} (\ell_2, \mathtt{r}_2) \cdots \in \mathsf{Runs}^\omega(\mathcal{M}'),$$

where

$$a_i' = (a_i, \mathtt{f}_i),$$
$$b_i' = (b_i, \mathtt{Y}_i).$$

If the induced play $\pi' = \mathsf{runToPlay}(\rho') = (a_1', b_1', \tau_1)(a_2', b_2', \tau_2) \cdots \in \mathbb{T}^\omega(A' \times B')$ is not in W_k^{A}, then Bob wins and we are done. Assume $\pi' \in W_k^{\mathsf{A}}$, and thus conditions 1–3 are satisfied. We argue that $\pi' \in W_{k,m}^{\mathsf{B}}$. The conditions 4 and 5 hold by construction. Aiming at demonstrating that 6 holds too, let $\mu_0 = \lambda \mathtt{x}.0$, $\tau_0 = 0$, and, for $i \geq 0$, let

$$\mu_{i+1} = \begin{cases} \mu_i & a_i = \square \quad \text{(improper round)} \\ (\mu_i)[\mathtt{Y}_i \mapsto \tau_{i+1}] & a_i \in A \quad \text{(proper round).} \end{cases} \tag{6.8}$$

Thus reset assignments μ_i are defined exactly as ν_i in (6.5) except that only proper requests are interpreted as clock resets. We claim that

Claim 6.9.

The region information $\mathtt{r}_i$ is consistent with μ_i:

$$\mathtt{r}_i = [\mu_i]_{\mathtt{X},m}.$$

Indeed, this is due to $\pi' \in W_k^{\mathsf{A}}$, and the fact that $\mathcal{M}'$ updates its stored region consistently with time elapse: at every round $\mathcal{M}'$ uses the successor region agreeing with the current fractional region submitted by Alice, and resets a set of clocks $\mathtt{Y}$ exactly when she plays a proper move responded by a move of the form $(b, \mathtt{Y}) \in B \times \mathcal{P}(\mathtt{X})$. Since an $\mathtt{x}$-request is submitted by $\mathcal{M}'$ only when $\hat{r} \models x \leq m{-}1$, condition 6 holds.

In order to show that Bob is winning, consider an $\mathcal{M}'$-conformant run ρ'. It suffices to show $\pi' = \mathsf{runToPlay}(\rho') \notin \phi^{-1}(W)$. Let the proper moves in ρ' be at indices $1 = i_1 < i_2 < \cdots$ ($i_1 = 1$ due to zero-starting). In particular, $\ell_{i_l} = \ell_i$ for $i_l \le i < i_{l+1}$. Consider the run

$$\rho = (\ell_0, \mu_0, 0) \xrightarrow{a_{i_1}, b_{i_1}, \tau_{i_1}} (\ell_{i_1}, \mu_{i_1}, \tau_{i_1}) \xrightarrow{a_{i_2}, b_{i_2}, \tau_{i_2}} (\ell_{i_2}, \mu_{i_2}, \tau_{i_2}) \cdots$$

Using claim 6.9 and the definition of $\mathcal{M}'$, one can prove by induction that ρ is an $\mathcal{M}$-conformant run in G. Since $\mathcal{M}$ is winning, the induced play $\pi = \mathsf{runToPlay}(\rho) = (a_{i_1}, b_{i_1}, \tau_{i_1})(a_{i_2}, b_{i_2}, \tau_{i_2}) \cdots \in \mathbb{T}^\omega(A \times B)$, satisfies $\pi \notin W$. Again by induction one can prove that $\pi = \phi(\pi')$. Hence $\phi(\pi') \notin W$ as required. □

Proof of lemma 6.8.

In what follows we restrict to plays satisfying W_k^{A}. For proving lemma 6.8, the converse of lemma 6.7, we need to understand the general shape of any possible untimed winning controller $\mathcal{M}' = (A', B', L', \ell'_0, \delta')$ in G'.

We say that such an $\mathcal{M}'$ is *complete* if its control locations are of the form $L' = L \times \mathsf{Reg}(\mathtt{X}, m) \times \mathsf{FReg}(\mathtt{X})$, $\ell'_0 = (\ell_0, \mathtt{r}_0, \mathtt{f}_0)$, and every $\mathcal{M}'$-conformant run is of the form

$$(\ell_0, \mathtt{r}_0, \mathtt{f}_0) \xrightarrow{(a_1, \mathtt{f}_1), (b_1, \mathtt{Y}_1), \tau_1} (\ell_1, \mathtt{r}_1, \mathtt{f}'_1) \xrightarrow{(a_2, \mathtt{f}_2), (b_2, \mathtt{Y}_2), \tau_2} (\ell_2, \mathtt{r}_2, \mathtt{f}'_2) \cdots, \tag{6.9}$$

where for each $i \ge 1$, the fractional region $\mathtt{f}'_i$ stored in a location agrees with the region $\mathtt{r}_i$, its domain $\mathsf{dom}(\mathtt{f}'_i) = \{\mathtt{x} \in \mathtt{X} \mid \text{there is an } \mathtt{x}\text{-request at time } \xi \text{ with } \tau_i - 1 < \xi \le \tau_i\}$, and $\mathtt{r}_i = [\tau_i - \mu_i]_{\mathtt{X},m}$ for the clock reset assignments μ_i as defined in (6.8). The proof has two steps, formulated in the claims below.

Claim 6.10. first step

If there is a winning untimed controller $\mathcal{M}' \in \mathsf{DTA}_0$ for Bob in G', then there is a winning complete one.

Indeed, it is not difficult to see that complete winning controllers suffice in G'. When Alice plays $a' = (a, \mathtt{f})$, the complete controller simulates $\mathcal{M}'$. Additionally, it uses the fractional region $\mathtt{f}$ and current region $\mathtt{r}$ to compute the next region $\mathtt{r}'$ (similarly as in the proof of lemma 6.7) and the next fractional region $\mathtt{f}'$. Let $\hat{\mathtt{r}} = \mathsf{succ}_{\mathtt{X},m}(\mathtt{r}, \mathtt{f})$, hence $\mathtt{f}$ agrees with $\hat{\mathtt{r}}$. Then, $\mathtt{r}' = \hat{\mathtt{r}}$ in improper moves, and in proper moves of the form $b' = (b, \mathtt{Y})$, let $\mathtt{r}' = \hat{\mathtt{r}}[\mathtt{Y} \mapsto 0]$. Let $\mathtt{f}''$ be restriction of $\mathtt{f}$ to $\mathsf{dom}(\mathtt{f}) \setminus \mathsf{integer}(\mathtt{f})$, and let $\mathsf{dom}(\mathtt{f}') = \mathsf{dom}(\mathtt{f}'') \cup \mathtt{Y}$ and $\mathtt{f}' = \mathtt{f}''[\mathtt{Y} \mapsto 0]$ (thus $\mathsf{dom}(\mathtt{f}')$ possibly increases in the case of proper move). This ensures that $\mathtt{f}'$ agrees with $\mathtt{r}'$ and $\mathsf{dom}(\mathtt{f}')$ contains all requested clocks.

It now remains to show that

Claim 6.11. second step

If there is a complete untimed winning controller $\mathcal{M}' \in \mathsf{DTA}_0$ for Bob in G' then there is a winning $\mathcal{M} \in \mathsf{DTA}_{k,m}$ for Bob in G.

Let $\mathcal{M}' = (A', B', L', \ell_0', \delta')$ be a winning complete controller in G' with $L' = L \times \mathsf{Reg}(\mathtt{X}, m) \times \mathsf{FReg}(\mathtt{X})$, $\ell_0' = (\ell_0, \mathtt{r}_0, \mathtt{f}_0)$, and update function of the form $\delta' : L' \times A' \to L' \times B'$. We define a winning k-clock m-constrained controller $\mathcal{M} = (A, B, L', \ell_0', \delta)$ in G over the same set of control locations L', and update function $\delta : L' \times A \times \mathsf{Reg}(\mathtt{X}, m) \to L' \times B \times 2^{\mathtt{X}}$. In order to define one step of δ (which corresponds to a proper move) we need to take many steps of δ' to skip all improper moves preceding the corresponding proper one. Let

$$\delta((\ell, \mathtt{r}, \mathtt{f}), a, \hat{\mathtt{r}}) = ((\ell'', \mathtt{r}'', \mathtt{f}''), b, \mathtt{Y}), \tag{6.10}$$

for $a \in A$, be recursively defined as follows:

1. In the base case, we have $\mathtt{r} \preceq \hat{\mathtt{r}}$ and $\mathtt{f}$ agrees with $\hat{\mathtt{r}}$ (as a special case we may have $\mathtt{r} = \hat{\mathtt{r}}$). We apply the transition function of $\mathcal{M}'$ and obtain directly the r.h.s. in (6.10) as $((\ell'', \mathtt{r}'', \mathtt{f}''), b, \mathtt{Y}) = \delta'((\ell, \mathtt{r}, \mathtt{f}), (a, \mathtt{f}))$ where $\mathtt{r}'' = \mathtt{r}[\mathtt{Y} \mapsto 0]$ and $\mathtt{f}''$ agrees with $\mathtt{r}''$.

2. In the next case, we have $\mathtt{r} \prec \hat{\mathtt{r}}$ and $\mathtt{f}$ does not agree with $\hat{\mathtt{r}}$. Let $\mathtt{f}'$ be the immediate successor of $\mathtt{f}$, and let $\delta'((\ell, \mathtt{r}, \mathtt{f}), (\square, \mathtt{f}')) = ((\ell', \mathtt{r}', \bar{\mathtt{f}}'), (\square, \_))$, where necessarily $\mathtt{r}' = \mathsf{succ}_{\mathtt{X},m}(\mathtt{r}, \mathtt{f}')$, and $\mathtt{r}'$ agrees with $\mathtt{f}'$. Then, we recursively define the r.h.s. in (6.10) as $((\ell'', \mathtt{r}'', \mathtt{f}''), b, \mathtt{Y}) = \delta((\ell', \mathtt{r}', \bar{\mathtt{f}}'), a, \hat{\mathtt{r}})$.

3. In any other case, $\hat{\mathtt{r}}$ is not a successor region of $\mathtt{r}$. Thanks to completeness (6.9), $\mathtt{r}$ is the region of the current clock valuation, and thus the controller can be defined arbitrarily because Bob is already winning, since Alice is losing due to violation of W_k^{A}.

The recursion above ends, and thus δ is well-defined, since there are only finitely many regions and $\prec$ is a strict total order on regions.

Consider an infinite $\mathcal{M}$-conformant run $\rho \in \mathsf{Runs}^\omega(\mathcal{M})$. By the definition of δ, there is a corresponding $\mathcal{M}'$-conformant run $\rho' \in \mathsf{Runs}^\omega(\mathcal{M})$ as in (6.9) where Alice in G' plays optimally (satisfying W_k^{A}), and ρ arises from ρ' by combining together adjacent sequences of improper moves: Let the proper moves in ρ' be at indices $1 = i_1 < i_2 < \cdots$. Then, ρ is of the form

$$\begin{aligned}\rho = ((\ell_0, \mathtt{r}_0, \mathtt{f}_0), \mu_0, 0) &\xrightarrow{a_1, b_1, \tau_{i_1}} ((\ell_{i_1}, \mathtt{r}_{i_1}, \mathtt{f}_{i_1}), \mu_{i_1}, \tau_{i_1}) \\ &\xrightarrow{a_2, b_2, \tau_{i_2}} ((\ell_{i_2}, \mathtt{r}_{i_2}, \mathtt{f}_{i_2}), \mu_{i_2}, \tau_{i_2}) \cdots ,\end{aligned}$$

where $a_j = \phi(a'_{i_j})$ and $b_j = \phi(b'_{i_j})$. Since Alice plays optimally when building ρ', the corresponding play $\pi' = \mathsf{runToPlay}(\rho') = (a'_1, b'_1, \tau_1)(a'_2, b'_2, \tau_2) \cdots$ is in W_k^{A}, and since $\mathcal{M}'$ is winning, $\pi' \in W_{k,m}^{\mathsf{B}}$ and $\pi' \notin \phi^{-1}(W)$. If the corresponding play $\pi = \mathsf{runToPlay}(\rho) = (a_1, b_1, \tau_{i_1})(a_2, b_2, \tau_{i_2}) \cdots$ in G was winning for Alice, which means $\pi \in W$, since $\phi(\pi') = \pi$ we would have $\pi' \in \phi^{-1}(W)$, a contradiction. $\square$

6.2.2 Case of unbounded constants in clock constraints

In this section we prove theorem 6.2, stating that the k-clock timed synthesis problem is computable, by reducing it to the 0-clock synthesis problem, which is computable by lemma 6.7. We build on the game defined in §6.2.1. Starting from a timed game

$G = G^{\mathbb{T}}_{A,B}(W)$ we define the timed game $G'' = G^{\mathbb{T}}_{A',B'}(W''_k)$, where the sets of actions A' and B' are as in (6.3), and the winning condition W''_k is defined as follows. Let $W^{\mathsf{B}}_k \subseteq (A' \cdot B' \cdot \mathbb{Q}_{\geq 0})^\omega$ be the set of plays where, for every clock $\mathtt{x} \in \mathtt{X}$, improper $\mathtt{x}$-request chains have finite lengths: $W^{\mathsf{B}}_k = \bigcup_{m \in \mathbb{N}} W^{\mathsf{B}}_{k,m}$. (In other words, $(A' \cdot B' \cdot \mathbb{Q}_{\geq 0})^\omega \setminus W^{\mathsf{B}}_k$ contains plays with an infinite improper $\mathtt{x}$-request chain, for some clock $\mathtt{x} \in \mathtt{X}$.) Then, W''_k is defined as $W'_{k,m}$ from (6.6), except that $W^{\mathsf{B}}_{k,m}$ is replaced by the weaker condition W^{B}_k (notice W''_k does not depend on m):

$$W''_k \;=\; W^{\mathsf{A}}_k \cap \Big(\phi^{-1}(W) \cup (A' \cdot B' \cdot \mathbb{Q}_{\geq 0})^\omega \setminus W^{\mathsf{B}}_k\Big). \tag{6.11}$$

Lemma 6.12.

There is a winning untimed controller for G'' if, and only if, there is some $m \in \mathbb{N}$ and a winning untimed controller for $G' = G^{\mathbb{T}}_{A',B'}(W'_{k,m})$.

Proof of lemma 6.12.

For the 'if' direction, we observe that $W''_k \subseteq W'_{k,m}$, for every $m \in \mathbb{N}$. Hence every winning untimed controller for G' is also winning for G''.

For the 'only if' direction, let $\mathcal{M}'' = (A', B', L, \ell_0, \delta)$ be an untimed winning controller in G''. Let $m = |A'| \cdot |L| + 1$. We claim that $\mathcal{M}''$ is also winning in $G' = G^{\mathbb{T}}_{A',B'}(W'_{k,m})$ for this choice of m.

Towards reaching a contradiction, suppose $\mathcal{M}''$ is losing in G'. An $\mathcal{M}''$-conformant run ρ in G' (or in G'') and its associated play π are of the form

$$\begin{aligned}\rho \;&= \ell_0(a'_1, b'_1, \tau_1, \ell_1)(a'_2, b'_2, \tau_2, \ell_2) \cdots \in \mathsf{Runs}^\omega(\mathcal{M}''), \text{ with } a'_i = (a_i, \mathtt{f}_i) \text{ and } b'_i = (b_i, \mathtt{Y}_i),\\ \pi \;&= \mathsf{runToPlay}(\rho) = (a'_1, b'_1, \tau_1)(a'_2, b'_2, \tau_2) \cdots \in \mathsf{Plays}(\mathcal{M}'').\end{aligned}$$

Let $\rho_i \in \mathsf{Runs}(\mathcal{M}'')$ be the finite prefix of ρ ending at $(a'_i, b'_i, \tau_i, \ell_i)$. Since $\mathcal{M}''$ is losing in G', some $\mathcal{M}''$-conformant play π above is in $W'_{k,m}$. Since $\mathcal{M}''$ is winning in G'', $\pi \notin \phi^{-1}(W)$, and thus $\pi \in W^{\mathsf{A}}_k \setminus W^{\mathsf{B}}_{k,m}$. This means that π contains an improper $\mathtt{x}$-request chain C of length m, for some clock $\mathtt{x} \in \mathtt{X}$. By the definition of m, there are indices $i < j$ such that the same controller memory repeats together with Alice's action $(a'_i, \ell_i) = (a'_j, \ell_j)$. In particular $\mathtt{f}_i = \mathtt{f}_j$. Since $\mathcal{M}''$ is deterministic and its action depends only on Alice's action a'_i and control location ℓ_i, a posteriori we have $b'_i = b'_j$ as well. Moreover, as consecutive timestamps in C are equal to the first one plus consecutive nonnegative integers, $\Delta = \tau_i - \tau_j \in \{1, \ldots, m-1\}$.

Consider the corresponding infix

$$\sigma = (a'_{i+1}, b'_{i+1}, \tau_{i+1}, \ell_{i+1}) \cdots (a'_j, b'_j, \tau_j, \ell_j)$$

of the run ρ. Since $\pi \in W'_{k,m}$, thanks to conditions 2 and 3 the fractional regions $\mathtt{f}_i = \mathtt{f}_j$ contain all tracked clocks, and they agree with the clock valuations ν_i and ν_j, respectively, as defined in (6.5). Let $\{\tau_i - 1 \leq \tau_{i_1} < \tau_{i_2} < \cdots < \tau_{i_l} < \tau_i\} = \{\tau_i - \nu_i(\mathtt{x}) \mid \mathtt{x} \in \mathsf{dom}(\mathtt{f}_i)\}$ be the timestamps corresponding to the last request of the clocks tracked at time τ_i, and likewise let $\{\tau_j - 1 \le \tau_{j_1} < \tau_{j_2} < \cdots < \tau_{j_{l'}} < \tau_j\} = \{\tau_j - \nu_j(\mathtt{x}) \mid \mathtt{x} \in \mathsf{dom}(\mathtt{f}_j)\}$. By assumption, $\mathtt{f}_i = \mathtt{f}_j$, and hence $l = l'$ and for $\mathtt{x} \in \mathsf{dom}(\mathtt{f}_i) = \mathsf{dom}(\mathtt{f}_j)$ and $1 \le h \le l$,

$$\tau_{i_h} = \tau_i - \nu_i(\mathtt{x}) \iff \tau_{j_h} = \tau_j - \nu_j(\mathtt{x}). \tag{6.12}$$

Moreover, since $\mathsf{integer}(\mathtt{f}_i) = \mathsf{integer}(\mathtt{f}_j)$, we have

$$\tau_{i_1} = \tau_i - 1 \iff \tau_{j_1} = \tau_j - 1. \tag{6.13}$$

Alice will win in G' by forcing a repetition of the infix σ *ad libitum*. In order to do so, we need to modify its timestamps. An *automorphism* of the structure $(\mathbb{Q}, \le, +1)$ is a monotonic bijection preserving integer differences, in the sense that $f(x+1) = f(x) + 1$ for every $x \in \mathbb{Q}$. Note that such an automorphism is uniquely defined by its action on any unit-length interval. We claim that there exists such an automorphism $f : \mathbb{Q} \to \mathbb{Q}$ mapping $\tau_i - 1$ to $\tau_j - 1$ (and hence forcedly also τ_i to τ_j), and each τ_{i_h} with $1 \le h \le l$ to $f(\tau_{i_h}) = \tau_{j_h}$.

This is indeed the case, by eqs. (6.12) and (6.13) all timestamps τ_{i_h}'s belong to the unit half-open interval $[\tau_i - 1, \tau_i)$ and likewise all timestamps τ_{j_h}'s belong to $[\tau_j - 1, \tau_j)$. We apply f to a timed word $\sigma \mapsto f(\sigma)$ by acting pointwise on timestamps. Consider the infinite run

$$\rho' \;=\; \rho_i \cdot \sigma \cdot f(\sigma) \cdot f(f(\sigma)) \cdots ;$$

it is $\mathcal{M}''$-conformant since the controller $\mathcal{M}''$ is deterministic, and thus behaves the same at every iteration. By construction, ρ' contains an infinite $\mathtt{x}$-request chain, and thus $\rho' \notin W_k^{\mathsf{B}}$.

It remains to argue that $\rho \in W_k^{\mathsf{A}}$ implies $\rho' \in W_k^{\mathsf{A}}$ as well. Let there be a non-cancelled $\mathtt{x}$-request at time τ_s in ρ'.

If $\tau_s < \tau_j - 1$, then this request must be satisfied at time $\tau_{s'} = \tau_s + 1 < \tau_j$, and thus already in $\rho_i \cdot \sigma$, which is the case since the latter is a prefix of $\rho \in W_k^{\mathsf{A}}$.

Now assume $\tau_j - 1 \le \tau_s < \tau_j$. Thus $\tau_s = \tau_{j_h}$ for some $1 \le h \le l$. By the definition of f, $f^{-1}(\tau_s) = \tau_{i_h} < \tau_j - 1$ and, thanks to the previous case, the request at τ_{i_h} is satisfied at $\tau_{i_h} + 1$ due to (*). By applying f we obtain $f(\tau_{i_h} + 1) = f(\tau_{i_h}) + 1 = \tau_s + 1$, and thus the request at time τ_s is satisfied at time $\tau_s + 1$ in $f(\sigma)$, as required.

The general argument for $\tau_j + n\Delta + d - 1 \le \tau_s < \tau_j + n\Delta + d$, where $n \ge 0$ and $0 \le d < \Delta$, is similar, using induction on n. $\square$

Proof of theorem 6.2.

Due to lemmas 6.7, 6.8 and 6.12, there is a winning untimed controller $\mathcal{M}''$ for G'' if, and only if there is some $m \in \mathbb{N}$ and a winning k-clock m-constrained controller $\mathcal{M}$ for G. Thus the k-clock timed synthesis problem reduces to the 0-clock 0-constrained timed synthesis problem, and the latter is computable thanks to lemma 6.13. $\square$

6.3 Elliminating time

The 0-clock timed synthesis problem is equivalent to untimed synthesis problem, which is computable thanks to the Büchi-Landweber result [15, Theorem 1′]:

Lemma 6.13.

The 0-clock synthesis problem is computable.

Proof of lemma 6.13.

Consider a timed synthesis game $G^{\mathbb{T}}_{A,B}(W)$ and let $W' = \mathsf{proj}_{\mathbb{T}}(W)$. Winning 0-register controllers in $G^{\mathbb{T}}_{A,B}(W)$ are in one-to-one correspondence with winning controllers in the corresponding untimed synthesis game with winning condition W'. Indeed, the update function $\delta : L \cdot A \cdot \mathsf{Reg}(k, m) \to L \cdot B \cdot 2^{\mathtt{X}}$ of a k-clock m-constrained controller $\mathcal{M}$ when $k = m = 0$ can equivalently be presented as a function of type $L \cdot A \to L \cdot B$ (which we take as the update function in the untimed controller $\mathcal{M}'$), and all functions of the latter type arise in this way.

If $\mathcal{M}$ is losing in $G^{\mathbb{T}}_{A,B}(W)$, then there is a $\mathcal{M}$-conformant run $\rho \in W$, and thus $\mathsf{proj}_{\mathbb{T}}(\rho)$ is a $\mathcal{M}'$-conformant run in W', showing that $\mathcal{M}'$ is losing in the corresponding untimed synthesis game.

On the other hand, let $\rho' \in W'$ be $\mathcal{M}'$-conformant. Since $\mathcal{M}$ does not look at the timestamps, we can choose them accordingly in order to find an $\mathcal{M}$-conformant timing thereof $\rho \in \mathsf{proj}_{\mathbb{T}}^{-1}(\rho') \cap W$.

Untimed synthesis is computable by theorem 2.30. □

Chapter 7

Solving deterministic membership problems

7.1 Decidability of DRA_1 membership for NRA_1

In this section we prove the decidability of deterministic membership for register automata:

Theorem 7.1. DRA_k membership

For every fixed $k \in \mathbb{N}$, the DRA_k membership problem is decidable for NRA_1 languages.

The technical development of this section will also serve as a preparation for the more involved case of timed automata from §§7.2.1 and 7.3. The key ingredient used in the proof of theorem 7.1 is the following characterisation of those NRA_1 languages which are also DRA_k languages. In particular, this characterisation provides a bound on the number of control locations of a DRA_k equivalent to a given NRA_1 (if any exists).

Lemma 7.2.

Let $\mathcal{A}$ be an NRA_1 with n control locations, and let $k \in \mathbb{N}$. The following conditions are equivalent:

1. $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$ for some DRA_k $\mathcal{B}$.
2. For every data word w, there is $S \subseteq \mathbb{A}$ of size at most k such that the left quotient $w^{-1}\mathcal{L}(\mathcal{A}) = \{v \mid w \cdot v \in \mathcal{L}(\mathcal{A})\}$ is S-invariant.
3. $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$ for some DRA_k $\mathcal{B}$ with at most $f(k,n) = (k+1)! \cdot 2^{n\cdot(k+1)}$ control locations.

Using the above lemma we derive a proof of theorem 7.1:

Proof of theorem 7.1.

Given an NRA_1 $\mathcal{A}$, the decision procedure enumerates all DRA_k $\mathcal{B}$ with at most $f(k,n)$ locations and checks whether $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$, using corollary 2.19. If no such DRA_k $\mathcal{B}$ is found, the procedure answers negatively. Note that the fact that $\mathcal{A}$ is NRA_1 is crucial here, since otherwise the equivalence check above would not be decidable. In fact, this is the only place where the one register restriction really plays a role: A generalisation of lemma 7.2 for $\mathcal{A}$ being a NRA_l for $l \geq 1$ could be stated and proved; however we prefer to avoid the additional notational complications of dealing with the general case since we need lemma 7.2 only in the case $l = 1$. □

Remark 7.3. complexity

The decision procedure for NRA_1 invokes the Ackermann subroutine to check equivalence between an NRA_1 and a candidate DRA. This is in a sense unavoidable, since we show in theorem 7.17 that the DRA_k membership problem is Ackermann-hard for NRA_1.

The proof of lemma 7.2 is presented below in §7.1.1. We remark that the lemma partially follows from the literature of automata theory in sets with atoms. For instance, the most interesting implication (2) $\Longrightarrow$ (3), even for $\mathcal{A}$ an arbitrary NRA_l with $l \geq 1$, minus the concrete bound $f(k, n)$ on control locations of $\mathcal{B}$, follows from the following known facts:

a) $\mathcal{L}(\mathcal{A})$ is recognised by a nondeterministic equivariant orbit-finite automaton [10, Theorem 5.11],

b) the assumption (2) implies that the Myhill-Nerode equivalence of L has orbit-finite index,

c) thus by [10, Theorem 5.14] $\mathcal{L}(\mathcal{A})$ is recognised by a deterministic orbit-finite automaton $\mathcal{B}'$, and

d) we can construct from $\mathcal{B}'$ some language-equivalent $\mathsf{DRA}_{k'}$ $\mathcal{B}$.

However, this would not yield

1) the fact that we can even take $k' = k$ (i.e., the number of registers of $\mathcal{B}$ can be taken to be the size k of the supports S in the assumption), and

2) the concrete bound $f(k, n)$ on the number of control locations of $\mathcal{B}$ (a computable such bound is necessary in the automata enumeration procedure in the proof of theorem 7.1).

For these reasons, we provide a full proof of the lemma.

7.1.1 Proof of lemma 7.2

Let us fix an NRA_1 $\mathcal{A} = (\mathtt{X}, \Sigma, L, L_{\mathsf{ini}}, L_{\mathsf{fin}}, \Delta)$ and $k \in \mathbb{N}$. Let $n = |L|$ be the number of control locations of $\mathcal{A}$. The implication (3) $\Longrightarrow$ (1) holds trivially. The implication (1) $\Longrightarrow$ (2) holds just because every left quotient $w^{-1}\mathcal{L}(\mathcal{A})$ is the same as $w^{-1}\mathcal{L}(\mathcal{B})$ by the assumption $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$ for a DRA_k $\mathcal{B}$, and, since $\mathcal{B}$ is deterministic, the latter quotient $w^{-1}\mathcal{L}(\mathcal{B})$ equals $\mathcal{L}(\mathcal{B}; c)$ for some configuration $c = (p, \mu)$. The latter is $\mu(\mathtt{X})$-invariant by lemma 2.20(3), and clearly $|\mu(\mathtt{X})| \leq k$. (Notice that $\mathcal{A}$ did not play a role here.)

It thus remains to prove the implication (2) $\Longrightarrow$ (3), which is the content of the rest of the section. Assuming (2), we are going to define a DRA_k $\mathcal{B}'$ with registers $\mathtt{X} = \{\mathtt{x}_1, \ldots, \mathtt{x}_k\}$

and with at most $f(k,n)$ locations such that $\mathcal{L}(\mathcal{B}') = \mathcal{L}(\mathcal{A})$. We start from the transition system $\mathcal{X}$ obtained by the finite powerset construction underlying the determinisation of $\mathcal{A}$. Next, after a series of language-preserving transformations, we will obtain a transition system isomorphic to the reachable part of $[\![\mathcal{B}']\!]$ for some DRA_k $\mathcal{B}'$. As the last step, we extract from this deterministic transition system a syntactic definition of $\mathcal{B}'$. This is achievable due to the invariance properties witnessed by the transition systems in the course of the transformation.

Macro-configurations

For simplicity, we will abuse the notation and write $c = (p,\alpha)$ for a configuration $c = (p, \{\mathtt{x}_1 \mapsto \alpha\})$ of $\mathcal{A}$, where $p \in L$ and $\alpha \in \mathbb{A} \cup \{\bot\}$. A *macro-configuration* is a (not necessarily finite) set X of configurations (p,α) of $\mathcal{A}$. We use the notation $\mathcal{L}(\mathcal{A};X) := \bigcup_{c\in X} \mathcal{L}(\mathcal{A};c)$.

Let $\mathsf{succ}_{\sigma,\alpha}(X) := \{c' \in [\![\mathcal{A}]\!] \mid c \xrightarrow{\sigma,\alpha} c' \text{ for some } c \in X\}$ be the set of successors of configurations in X which can be reached by reading $(\sigma,\alpha) \in \Sigma \times \mathbb{A}$. We define a deterministic transition system $\mathcal{X}$ consisting of the macro-configurations reachable in the course of determinisation of $\mathcal{A}$. Let $\mathcal{X}$ be the smallest set of macro-configurations and transitions such that

- ▶ $\mathcal{X}$ contains the initial macro-configuration: $X_0 = \{(p,\bot) \mid p \in L_{\mathsf{ini}}\} \in \mathcal{X}$;
- ▶ $\mathcal{X}$ is closed under successor: for every $X \in \mathcal{X}$ and $(\sigma,\alpha) \in \Sigma \times \mathbb{A}$, there is a transition $X \xrightarrow{\sigma,\alpha} \mathsf{succ}_{\sigma,\alpha}(X)$ in $\mathcal{X}$.

Due to the fact that $[\![\mathcal{A}]\!]$ is finitely branching, i.e. $\mathsf{succ}_{\sigma,\alpha}(\{c\})$ is finite for every fixed (σ,α), all macro-configurations $X \in \mathcal{X}$ are finite. Let the final configurations of $\mathcal{X}$ be $F_{\mathcal{X}} = \{X \in \mathcal{X} \mid X \cap F \neq \varnothing\}$ where $F \subseteq [\![\mathcal{A}]\!]$ is the set of final configurations of $\mathcal{A}$.

Claim 7.4.

$\mathcal{L}(\mathcal{A};X) = \mathcal{L}(\mathcal{X};X)$ for every $X \in \mathcal{X}$. In particular $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{X};X_0)$.

For a macro-configuration X we write $\mathsf{Val}(X) := \{\alpha \in \mathbb{A} \mid \exists p.(p,\alpha) \in X\}$ to denote the set of atoms appearing in X.

Pre-states

By assumption (2), for every macro-configuration $X \in \mathcal{X}$, $\mathcal{L}(\mathcal{A};X)$ is S-invariant for some S of size at most k, but the macro-configuration X itself needs not be S-invariant in general. Indeed, a finite macro-configuration $X \in \mathcal{X}$ is S-invariant if, and only if, $\mathsf{Val}(X) \subseteq S$, which is impossible in general when X is arbitrarily large while the size of S is bounded (by k). Intuitively, in order to assure S-invariance we will replace X by its S-closure $\mathsf{closure}_S(X)$ (recall fact 2.13).

The *least support* of a set with atoms X is the least $S \subseteq \mathbb{A}$ w.r.t. set inclusion such that X is S-invariant. In the case of equality atoms every set with atoms has the least support, which is moreover finite (see [11, Cor. 9.4] or [10, Thm. 6.1]). By assumption, the least finite support of every macro-configuration X in $\mathcal{X}$ has size at most k.

A *pre-state* is a pair $Y = (X, S)$, where X is a macro-state whose least finite support is S. Thus X is S-invariant which, together with the fact that S has size at most k, implies that there are only finitely many pre-states up to automorphism. We define the deterministic transition system $\mathcal{Y}$ as the smallest set of pre-states and transitions between them such that:

- $\mathcal{Y}$ contains the initial pre-state: $Y_0 = (X_0, \varnothing) \in \mathcal{Y}$;
- $\mathcal{Y}$ is closed under the closure of successor: For every $(X, S) \in \mathcal{Y}$ and $(\sigma, \alpha) \in \Sigma \times \mathbb{A}$, the pre-state (X', S') is in $\mathcal{Y}$ together with transition $(X, S) \xrightarrow{\sigma,\alpha} (X', S')$, where S' is the least finite support of the language $L' = (\sigma, \alpha)^{-1}\mathcal{L}(\mathcal{A}; X) = \mathcal{L}(\mathcal{A}; \mathsf{succ}_{\sigma,\alpha}(X))$, and $X' = \mathsf{closure}_{S'}(\mathsf{succ}_{\sigma,\alpha}(X))$.

Example 7.5.

Suppose that $k = 3$, a successor of some macro-configuration X has the shape $\mathsf{succ}_{\sigma,\alpha_1}(X) = \{(p, \alpha_1), (q, \alpha_1), (r, \alpha_2), (s, \alpha_3)\}$ and the least finite support S' of L' is $\{\alpha_1, \alpha_3\}$, where $\alpha_1, \alpha_2, \alpha_3 \in \mathbb{A}$ are pairwise-different. Then $X' = \{(p, \alpha_1), (q, \alpha_1)\} \cup \{(r, \alpha) \mid \alpha \in \mathbb{A} \setminus \{\alpha_1, \alpha_3\}\} \cup \{(s, \alpha_3)\}$.

By assumption, L' is T-invariant for some $T \subseteq \mathbb{A}$ with $|T| \leq k$. Since X is S-invariant, L' is also $(S \cup \{\alpha\})$-invariant. By the least finite support property of equality atoms, finite supports are closed under intersection, and hence $S' \subseteq (S \cup \{\alpha\}) \cap T$, which implies $|S'| \leq k$.

By lemma 2.20 we deduce:

Claim 7.6. invariance of $\mathcal{Y}$

For every two transitions $(X_1, S_1) \xrightarrow{\sigma,\alpha_1} (X_1', S_1')$ and $(X_2, S_2) \xrightarrow{\sigma,\alpha_2} (X_2', S_2')$ in $\mathcal{Y}$ and an automorphism π, if $\pi(X_1) = X_2$ and $\pi(S_1) = S_2$ and $\pi(\alpha_1) = \alpha_2$, then we have $\pi(X_1') = X_2'$ and $\pi(S_1') = S_2'$.

Let the final configurations of $\mathcal{Y}$ be $F_{\mathcal{Y}} = \{(X, S) \in \mathcal{Y} \mid X \cap F \neq \varnothing\}$. By induction on the length of data words it is easy to show:

Claim 7.7.

$\mathcal{L}(\mathcal{X}; X_0) = \mathcal{L}(\mathcal{Y}; Y_0)$.

States

We now introduce *states*, which are designed to be in one-to-one correspondence with configurations of the forthcoming DRA_k $\mathcal{B}'$. Intuitively, a state differs from a pre-state (X, S) only by allocating the values from (some superset of) S into k registers. Thus, while a pre-state contains a set S, the corresponding state contains a register assignment $\mu : \mathbb{A}_\perp^{\mathtt{X}}$ with image $\mu(\mathtt{X}) \supseteq S$.

Let $\mathtt{X} = \{\mathtt{x}_1, \ldots, \mathtt{x}_k\}$ be a set of k registers. A *state* is a pair $Z = (X, \mu)$, where X is a macro-configuration, $\mu : \mathbb{A}_\perp^{\mathtt{X}}$ is a register assignment, and X is $\mu(\mathtt{X})$-invariant. Thus every state (X, μ) determines uniquely a corresponding pre-state $\rho(X, \mu) = (X, S)$ where $S \subseteq \mu(\mathtt{X})$ is the least finite support of X.

Example 7.8.

We continue example 7.5. States corresponding to the pre-state (X', S') feature the macro-configuration X', but can have different register valuations. One of them is (X', μ'), where $\mu' = \{\mathtt{x}_1 \mapsto \alpha_1, \mathtt{x}_2 \mapsto \alpha_3, \mathtt{x}_3 \mapsto \alpha_1\}$.

We now define a deterministic transition system $\mathcal{Z}$. Its states are all those (X, μ) satisfying $\rho(X, \mu) \in \mathcal{Y}$, and transitions are determined as follows: $\mathcal{Z}$ contains a transition $(X, \mu) \xrightarrow{\sigma,\alpha} (X', \mu')$ if $\mathcal{Y}$ contains the corresponding transition $\rho(X, \mu) \xrightarrow{\sigma,\alpha} \rho(X', \mu') = (X', S')$, and $\mu' = \mu[\mathtt{Y} \mapsto \alpha]$, where

$$\mathtt{Y} \;=\; \{\mathtt{x}_i \in \mathtt{X} \mid \mu(\mathtt{x}_i) \notin S' \text{ or } \mu(\mathtt{x}_i) = \mu(\mathtt{x}_j) \text{ for some } j > i\}. \tag{7.1}$$

The equation (7.1) defines a deterministic update policy[1] of the register assignment μ that amounts to updating with the current input atom α all registers $\mathtt{x}_i$ whose value is either no longer needed (because $\mu(\mathtt{x}_i) \notin S'$), or is shared with some other register x_j, for $j > i$ and is thus redundant. It is easy to see that the above register update policy guarantees that $S' \subseteq \mu'(\mathtt{X}) \subseteq S' \cup \{\alpha\}$. Using claim 7.6 we derive:

Claim 7.9. invariance of $\mathcal{Z}$

For every two transitions $(X_1, \mu_1) \xrightarrow{\sigma,\alpha_1} (X_1', \mu_1')$ and $(X_2, \mu_2) \xrightarrow{\sigma,\alpha_2} (X_2', \mu_2')$ in $\mathcal{Z}$ and an automorphism π, if $\pi(X_1) = X_2$ and $\pi \circ \mu_1 = \mu_2$ and $\pi(\alpha_1) = \alpha_2$, then we have $\pi(X_1') = X_2'$ and $\pi \circ \mu_1' = \mu_2'$.

Let the initial state be $Z_0 = (X_0, \lambda \mathtt{x}.\bot)$, and let final states be $F_{\mathcal{Z}} = \{(X, \mu) \in \mathcal{Z} \mid X \cap F \neq \varnothing\}$. By induction on the length of data words one proves:

Claim 7.10.

$\mathcal{L}(\mathcal{Y}; Y_0) = \mathcal{L}(\mathcal{Z}; Z_0)$.

In the sequel we restrict $\mathcal{Z}$ to states reachable from Z_0.

Orbits of states

Recall that the action of automorphisms on macro-configurations and reset-point assignments is extended to states as $\pi(X, \mu) = (\pi(X), \pi \circ \mu)$, and that the orbit of a state Z is defined as $\mathsf{orbit}(Z) = \{\pi(Z) \mid \pi \in \mathsf{Aut}(\mathbb{A})\}$.

While a state is designed to correspond to a configuration of the forthcoming DRA$_k$ $\mathcal{B}'$, its orbit is designed to play the role of control location of $\mathcal{B}'$. We therefore need to prove that the set of orbits $\{\mathsf{orbit}(Z) \mid Z \in \mathcal{Z}\}$ is finite and its size is bounded by $f(k, n)$.

Let M_k denote the number of orbits of register valuations $\mathbb{A}_\bot^{\mathtt{X}}$, which is the same as the number of orbits of k-tuples $\mathbb{A}_\bot^k$. In case of equality atoms we have $M_k \leq (k+1)!$ Indeed, at the first position there are two possibilities, $\bot$ or an atom; at the second position there are at most three possibilities: $\bot$, the same atom at the one at the first position, or a fresh atom; and so on, until the last position where there are at most $k+1$ possibilities.

[1] There are in general many correct deterministic update policies, but for our purposes it suffices to define one such deterministic update policy.

Every l-element subset $S = \{\alpha_1, \alpha_2, \dots, \alpha_l\} \subseteq \mathbb{A}$ of atoms induces, in the case of equality atoms, exactly $l + 1$ different S-orbits of atoms:

$$\mathsf{Orbits}_S(\mathbb{A}) \;=\; \{\mathsf{orbit}_S(\alpha) \mid \alpha \in \mathbb{A}\} \;=\; \{\{\alpha_1\}, \{\alpha_2\}, \dots, \{\alpha_l\}, \mathbb{A} - S\}.$$

Therefore, each $S \subseteq \mathbb{A}$ of size at most k induces at most $N_k := k+1$ different S-orbits of atoms.

Consider a state $Z = (X, \mu)$ and let $S = \mu(\mathtt{X})$. We define the characteristic function $\mathsf{char}_Z : \mathsf{Orbits}_S(\mathbb{A}) \to \mathcal{P}(L)$ as follows:

$$\mathsf{char}_Z(o) = \{l \in L \mid (l, \alpha) \in X \text{ for some } \alpha \in o\}.$$

Since X is S-invariant, the choice of the atom $\alpha \in o$ is irrelevant, and we conclude:

Claim 7.11.

Every state $Z = (X, \mu)$ is uniquely determined by its register valuation μ and by the characteristic function char_Z.

Claim 7.12.

The size of $\{\mathsf{orbit}(Z) \mid Z \in \mathcal{Z}\}$ is at most $M_k \cdot 2^{n \cdot N_k}$.

Proof of claim 7.12.

We show that there are at most $M_k \cdot (2^n)^{N_k}$ different orbits of states. Consider two states $Z = (X, \mu)$ and $Z' = (X', \mu)$ and let $S = \mu(\mathtt{X})$ and $S' = \mu'(\mathtt{X})$. Suppose that the register valuations μ and μ' are in the same orbit: $\pi \circ \mu = \mu'$ for some automorphism π. Thus $\pi(S) = S'$, and moreover π induces a bijection $\widetilde{\pi}$ between $\mathsf{Orbits}_S(\mathbb{A})$ and $\mathsf{Orbits}_{S'}(\mathbb{A})$. Note that, once S is fixed, there are at most $(2^n)^{N_k}$ possible characteristic functions of Z, and likewise for S' and Z'. Supposing further that the characteristic functions agree, i.e., satisfy $\mathsf{char}_Z = \mathsf{char}_{Z'} \circ \widetilde{\pi}$, using claim 7.11 we derive $\pi(Z) = Z'$, i.e., Z and Z' are in the same orbit. Therefore, since the number of orbits of register valuations μ, μ' is at most M_k, and for each such orbit the number of different characteristic functions is at most $(2^n)^{N_k}$, the number of different orbits of states is bounded as required. □

For future use we observe that every state is uniquely determined by its register valuation and its orbit:

Claim 7.13.

Let $Z = (X, \mu)$ and $Z' = (X', \mu)$ be two states in $\mathcal{Z}$ with the same register valuation. If $\pi(X) = X'$ and $\pi \circ \mu = \mu$ for some automorphism π then $X = X'$.

Proof of claim 7.13.

Indeed, X is $\mu(X)$-invariant and hence $\pi(X) = X$, which implies $X = X'$. □

In the terminology of automata in sets with atoms, we have proved that $\mathcal{Z}$ is a *deterministic orbit-finite automaton* (c.f. [10, Sec. 5.2]) for a definition), with the concrete bound on the number of orbits given by claim 7.12.

Construction of the DRA

As the last step we define a $\textsf{DRA}_k$ $\mathcal{B}' = (\mathtt{X}, \Sigma, L', \{o_0\}, L'_F, \Delta')$ such that the reachable part of $[\![\mathcal{B}']\!]$ is isomorphic to $\mathcal{Z}$. Let locations $L' = \{\mathsf{orbit}(Z) \mid Z \in \mathcal{Z}\}$ be the orbits of states from $\mathcal{Z}$, the initial location be the orbit o_0 of Z_0, and final locations $L'_F = \{\mathsf{orbit}(Z) \mid Z \in F_{\mathcal{Z}}\}$ be orbits of final states. Let each transition $Z = (X, \mu) \xrightarrow{\sigma,\alpha} (X', \mu') = Z'$ in $\mathcal{Z}$ induce a transition rule in $\mathcal{B}'$

$$(o, \sigma, \psi, \mathtt{Y}, o') \in \Delta' \tag{7.2}$$

where $o = \mathsf{orbit}(Z)$, $o' = \mathsf{orbit}(Z')$, $\mathtt{Y} = \{\mathtt{x} \in \mathtt{X} \mid \mu'(\mathtt{x}) = \alpha\}$, and the constraint $\psi(\mathtt{x}_1, \ldots, \mathtt{x}_k, \mathtt{y})$ defines the orbit of $(\mu(\mathtt{x}_1), \ldots, \mu(\mathtt{x}_k), \alpha)$ (here we rely on claim 2.14). We argue that the automaton $\mathcal{B}'$ is deterministic:

Claim 7.14.

Suppose that $\mathcal{B}'$ has two transition rules $o \xrightarrow{\sigma,\psi_1,\mathtt{Y}_1} o'_1$ and $o \xrightarrow{\sigma,\psi_2,\mathtt{Y}_2} o'_2$ with the same source location o and jointly satisfiable constraints ($[\![\psi_1 \wedge \psi_2]\!] \neq \varnothing$). Then the target locations are equal ($o'_1 = o'_2$), and the same registers are updated ($\mathtt{Y}_1 = \mathtt{Y}_2$).

Proof of claim 7.14.

Since the constraints are jointly satisfiable, both transition rules are enabled in some configuration $c = (o, \mu)$ and for some input atom $\alpha \in \mathbb{A}$. By claim 7.13, c determines a corresponding state $Z = (X, \mu)$ with $o = \mathsf{orbit}(Z)$ and, since the system $\mathcal{Z}$ is deterministic, both transition rules are induced by a common transition $(X, \mu) \xrightarrow{\sigma,\alpha} (X', \mu')$ in $\mathcal{Z}$. This in turn implies $o_1 = o_2$ and $\mathtt{Y}_1 = \mathtt{Y}_2$, as required. $\square$

Claim 7.15.

$\mathcal{Z}$ is isomorphic to the reachable part of $[\![\mathcal{B}']\!]$.

Proof of claim 7.15.

For a state $Z = (X, \mu)$, let $\iota(Z) = (\mathsf{orbit}(Z), \mu)$. Let $\mathcal{Z}'$ denote the reachable part of $[\![\mathcal{B}']\!]$. By claim 7.13, the mapping $\iota(\_)$ is a bijection between $\mathcal{Z}$ and its image $\iota(\mathcal{Z}) \subseteq [\![\mathcal{B}']\!]$. We aim at proving $\iota(\mathcal{Z}) = \mathcal{Z}'$.

By the very definition (7.2), the image $\iota(\mathcal{Z})$ is a subsystem of $\mathcal{Z}'$: $\iota(\mathcal{Z}) \subseteq \mathcal{Z}'$. For the converse inclusion, recall that $\mathcal{Z}$ is total: for every $(\sigma_1, \alpha_1) \ldots (\sigma_n, \alpha_n) \in (\Sigma \times \mathbb{A})^*$, there is a sequence of transitions $(X_0, \mu_0) \xrightarrow{\sigma_1,\alpha_1} \cdots \xrightarrow{\sigma_n,\alpha_n} (X_n, \mu_n)$ in $\mathcal{Z}$. Therefore $\iota(\mathcal{Z})$ is total too and, since $\mathcal{Z}'$ is deterministic and reachable, the subsystem $\iota(\mathcal{Z}) \subseteq \mathcal{Z}'$ necessarily equals $\mathcal{Z}'$. $\square$

Claims 7.4, 7.7, 7.10 and 7.15 jointly imply $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B}')$, which completes the proof of lemma 7.2.

7.1.2 Other atoms

The proof of theorem 7.1 straightforwardly generalises to any relational structure of atoms $\mathbb{A}$ satisfying the following conditions:

- $\mathbb{A}$ is homogeneous [47];
- $\mathbb{A}$ preserves well-quasi orders (WQO): finite induced substructures of $\mathbb{A}$ labelled by elements of an arbitrary WQO, ordered by label-respecting embedding, are again a WQO (for details we refer the reader to [69, item (A3), Sect.5]);
- $\mathbb{A}$ is effective: it is decidable, if a given finite structure over the vocabulary of $\mathbb{A}$ is an induced substructure thereof;
- $\mathbb{A}$ has the least finite support property.

As an example, the structure of densely ordered atoms $\mathbb{A} = (\mathbb{Q}, \leq)$ satisfies all the conditions and hence theorem 7.1 holds for register automata over this structure of atoms.

We briefly discuss the adjustments needed. The syntax of constraints (2.2) is extended by adding atomic constraints for all relations in $\mathbb{A}$, and claim 2.14 holds by homogeneity of $\mathbb{A}$. The decision procedure checking equivalence of an NRA_1 and a DRA from corollary 2.19, invoked in the proof of theorem 7.1, works assuming that $\mathbb{A}$ preserves WQO and is effective. The least finite support assumption is required in the definition of pre-states. Finally, again due to homogeneity of $\mathbb{A}$ the bounds M_k and N_k used in claim 7.12 are finite (but dependent on $\mathbb{A}$).

7.2 Lower bounds for NRA

The NRA *universality problem* asks whether a given NRA $\mathcal{A}$ recognises every data word $\mathcal{L}(\mathcal{A}) = (\Sigma \times \mathbb{A})^*$. All the lower bounds in this section leading to undecidability and hardness results for the DRA_k membership problem are obtained by a reduction from the universality problem for corresponding classes of data languages.

Lemma 7.16. c.f. [46, Theorem 1]

Let $k \in \mathbb{N}$ and let $\mathcal{Y}$ be a class of invariant data languages that

1. contains all the DRA_0 languages,
2. is closed under union and concatenation, and
3. contains some non-DRA_k language.

The universality problem for data languages in $\mathcal{Y}$ reduces in polynomial time to the DRA_k membership problem for data languages in $\mathcal{Y}$.

Proof of lemma 7.16.

Let $L \in \mathcal{Y}$ be a data language over a finite alphabet Σ. We show that universality of L reduces to DRA_k membership. Thanks to the last assumption, let $M \in \mathcal{Y}$ be a data language over some finite alphabet Γ which is not recognised by any DRA_k. Consider the following language over the extended alphabet $\Sigma' = \Sigma \cup \Gamma \cup \{\$\}$:

$$N \;:=\; L \cdot (\{\$\} \times \mathbb{A}) \cdot (\Gamma \times \mathbb{A})^* \,\cup\, (\Gamma \times \mathbb{A})^* \cdot (\{\$\} \times \mathbb{A}) \cdot M,$$

where $\$ \notin \Sigma \cup \Gamma$ is a fixed fresh alphabet symbol. Since $\mathcal{Y}$ contains the universal language, by its closure properties the language N belongs to $\mathcal{Y}$. We conclude by proving the following equivalence:

$$L = (\Sigma \times \mathbb{A})^* \quad \text{if, and only if,} \quad N \text{ is recognised by a } \mathsf{DRA}_k.$$

For the "only if" direction, if L is universal, then $N = (\Sigma \times \mathbb{A})^* \cdot (\{\$\} \times \mathbb{A}) \cdot (\Sigma \times \mathbb{A})^*$ is clearly recognised by a DRA_k. For the "if" direction suppose, towards reaching a contradiction, that N is recognised by a DRA_k $\mathcal{A}$ but L is not universal. Choose an arbitrary data word $w \notin L$ over Σ and consider an arbitrary extension $u = w \cdot (\$, \alpha)$ of w by one letter. Since $\$$ does not belong to the finite alphabet $\Sigma \cup \Gamma$, the left quotient $u^{-1}N = \{v \mid uv \in N\}$ equals M. Let (p, μ) be the configuration reached by $\mathcal{A}$ after reading u, which thus recognises $\mathcal{L}(p, \mu) = M$. Since M is invariant as a language in $\mathcal{Y}$, M is a DRA_k language, which is a contradiction. □

From lemma 7.16 we immediately obtain the undecidability and hardness results for the DRA_k membership problem, which we now recall.

Theorem 7.17. undecidability and hardness for DRA_k membership

Fix a $k \geq 0$. The DRA_k membership problem is:

1. undecidable for NRA_2,
2. undecidable for NRA_1^g (NRA_1 with guessing), and
3. not primitive recursive (Ackermann-hard) for NRA_1.

Proof of theorem 7.17.

For the first point, consider the class $\mathcal{Y}$ consisting of all the NRA_2 languages. Clearly this class contains all DRA_0 languages and it is closed under union and concatenation. Thanks to example 2.17 we know that there are NRA_1 (and thus NRA_2) languages which are not DRA languages. Thus the conditions of lemma 7.16 are satisfied and the universality problem for NRA_2 reduces in polynomial time to the DRA_k membership problem for NRA_2. Since the former problem is undecidable [40, Theorem 5.4], undecidability of the latter one follows. For the other two points we can proceed in an analogous way, by using the fact that the universality problem is undecidable for NRA_1^g (NRA_1 with guessing) [10, Exercise 9], and not primitive recursive for NRA_1 [40, Theorem 5.2]. □

7.2.1 Invariance of timed automata

A fundamental tool used below is invariance properties of timed languages recognised by NTA with respect to timed automorphisms. In this section we establish these properties, as extension of analogous properties of register automata. We also prove a timed analog of the least support property, and relate regions to orbits of configurations.

Recall that timed automorphisms are monotonic bijections $\mathbb{Q} \to \mathbb{Q}$ that preserve integer differences. A timed automorphism π acts on input letters in Σ as the identity, $\pi(a) = a$, and is extended point-wise to timed words $\pi((\sigma_1, \tau_1) \ldots (\sigma_n, \tau_n)) = (\sigma_0, \pi(\tau_1)) \ldots (\sigma_n, \pi(\tau_n))$, configurations $\pi(p, \mu, \tau_0) = (p, \pi \circ \mu, \pi(\tau_0))$, transitions $\pi(c \xrightarrow{\sigma,\tau} c') = \pi(c) \xrightarrow{\sigma,\pi(\tau)} \pi(c')$, and sets X thereof $\pi(X) = \{\pi(x) \mid x \in X\}$.

Remark 7.18.

In considerations about timed automata we restrict to nonnegative rationals, while a timed automorphism π can in general take a nonnegative rational $\tau \geq 0$ to a negative one. In the sequel whenever we write $\pi(x)$, for x being any object like a timestamp, a configuration, a timed word, etc., we always implicitly assume that π is well-defined on x, i.e., yields a timestamp, a configuration, a timed word, etc. In other words, for invariance properties we restrict to those timed automorphisms that preserve nonnegativeness of all the involved timestamps.

Let $S \subseteq \mathbb{Q}_{\geq 0}$. An *$S$-timed automorphism* is a timed automorphism such that $\pi(\tau) = \tau$ for all $\tau \in S$. Let Π_S denote the set of all S-timed automorphisms, and let $\Pi = \Pi_\varnothing$. A set X is *S-invariant* if $\pi(X) = X$ for every $\pi \in \Pi_S$; equivalently, for every $\pi \in \Pi_S$, $x \in X$ if, and only if $\pi(x) \in X$. A set X is *invariant* if it is S-invariant with $S = \varnothing$. The following three facts express some basic invariance properties.

Fact 7.19.

The timed transition system $[\![\mathcal{A}]\!]$ is invariant.

Proof of fact 7.19.

Suppose $c = (p, \mu, \tau_0) \xrightarrow{\sigma,\tau} (p', \mu', \tau) = c'$ due to some transition rule of $\mathcal{A}$ whose clock constraint φ compares values of clocks $\mathtt{x}$, i.e., the differences $\tau - \mu(\mathtt{x})$, to integers. Since a timed automorphism π preserves integer distances, the same clock constraint is satisfied in $\pi(c) = (p, \pi \circ \mu, \pi(\tau_0))$, and therefore the same transition rule is applicable yielding the transition $(p, \pi \circ \mu, \pi(\tau_0)) \xrightarrow{\sigma,\pi(\tau)} (p, \pi \circ \mu', \pi(\tau)) = \pi(c')$. $\square$

By unrolling the definition of invariance in the previous fact, we obtain that the set of configurations is invariant, the set of transitions $\to$ is invariant, and that the set of final configurations F is invariant.

Fact 7.20. Invariance of the language semantics

The function $c \mapsto \mathcal{L}(\mathcal{A}; c)$ from $[\![\mathcal{A}]\!]$ to languages is invariant, i.e., for all timed automorphisms π, $\mathcal{L}(\mathcal{A}; \pi(c)) = \pi(\mathcal{L}(\mathcal{A}; c))$.

Proof of fact 7.20.

Consider a timed automorphism π and an accepting run of $\mathcal{A}$ over a timed word $w = (\sigma_1, \tau_1) \ldots (\sigma_n, \tau_n) \in \mathbb{T}_{\geq \tau_0}(\Sigma)$ starting in $c = (p, \mu, \tau_0)$:

$$(p, \mu, \tau_0) \xrightarrow{\sigma_1, \tau_1} \cdots \xrightarrow{\sigma_n, \tau_n} (q, \nu, \tau_n),$$

After σ_i is read, the value of each clock is either the difference $\tau_i - \mu(\mathtt{x})$ for some $1 \leq i \leq n$ and clock $\mathtt{x} \in \mathtt{X}$, or the difference $\tau_i - \tau_j$ for some $1 \leq j \leq i$. Likewise is the difference of values of any two clocks. Thus clock constraints of transition rules used in the run compare these differences to integers. As timed automorphism π preserves integer differences, by executing the same sequence of transition rules we obtain the run over $\pi(w)$ starting in $\pi(c) = (p, \pi{\circ}\mu, \pi(\tau_0))$:

$$(p, \pi{\circ}\mu, \pi(\tau_0)) \xrightarrow{\sigma_1, \pi(\tau_1)} \cdots \xrightarrow{\sigma_n, \pi(\tau_n)} (q, \pi{\circ}\nu, \pi(\tau_n)),$$

also accepting as it ends in the same location q. As $w \in \mathbb{T}(\Sigma)$ can be chosen arbitrarily, we have thus proved one of inclusions, namely

$$\pi(\mathcal{L}(\mathcal{A}; p, \mu, \tau_0)) \ \subseteq \ \mathcal{L}(\mathcal{A}; p, \pi{\circ}\mu, \pi(\tau_0)).$$

The other inclusion follows from the latter one applied to π^{-1} and $\mathcal{L}(\mathcal{A}; p, \pi{\circ}\mu, \pi(\tau_0))$:

$$\pi^{-1}(\mathcal{L}(\mathcal{A}; p, \pi{\circ}\mu, \pi(\tau_0))) \ \subseteq \ \mathcal{L}(\mathcal{A}; p, \pi^{-1}{\circ}\pi{\circ}\mu, \pi^{-1}(\pi(\tau_0))) \ = \ \mathcal{L}(\mathcal{A}; p, \mu, \tau_0).$$

The two implications prove the equality. $\square$

Fact 7.21. **Invariance of the language of a configuration**

The language $\mathcal{L}(\mathcal{A}; p, \mu, \tau_0)$ is $(\mu(\mathtt{X}) \cup \{\tau_0\})$-invariant. Moreover, if $\mathcal{A}$ is always resetting, then $\mathcal{L}(\mathcal{A}; p, \mu, \tau_0)$ is $\mu(\mathtt{X})$-invariant.

Proof of fact 7.21.

This is a direct consequence of the invariance of semantics. Indeed, for every $(\mu(\mathtt{X}) \cup \{\tau_0\})$-timed automorphism π the configurations $c = (p, \mu, \tau_0)$ and $\pi(c) = (p, \pi{\circ}\mu, \pi(\tau_0))$ are equal, hence their languages $\mathcal{L}(\mathcal{A}; c)$ and $\mathcal{L}(\mathcal{A}; \pi(c))$, the latter equal to $\pi(\mathcal{L}(\mathcal{A}; c))$ by fact 7.20, are equal too. Thus, $L = \pi(L)$. Finally, if $\mathcal{A}$ is always resetting, then $\tau_0 \in \mu(\mathtt{X})$, from which the second claim follows. $\square$

Since timed automorphisms preserve integer differences, only the fractional parts of elements of $S \subseteq \mathbb{Q}_{\geq 0}$ matter for S-invariance, and hence it makes sense to restrict to subsets of the half-open interval $[0, 1)$. Let $\mathsf{fract}(S) = \{\mathsf{fract}(x) \mid x \in S\} \subseteq [0, 1)$ stand for the set of fractional parts of elements of S. The following lemma shows that, modulo the irrelevant integer parts, there is always the least set S witnessing S-invariance (c.f. the least support property of, e.g., equality atoms).

Lemma 7.22.

For finite subsets $S, S' \subseteq \mathbb{Q}_{\geq 0}$, if a timed language L is both S-invariant and S'-invariant, then it is also S''-invariant as long as $\mathsf{fract}(S'') = \mathsf{fract}(S) \cap \mathsf{fract}(S')$.

Proof of lemma 7.22.

Let L be an S- and S'-invariant timed language, and let $F = \mathsf{fract}(S)$ and $F' = \mathsf{fract}(S')$. We prove that L is an $(F \cap F')$-invariant subset of $\mathbb{T}(\Sigma)$. Consider two timed words $w, w' \in \mathbb{T}(\Sigma)$ such that $w' = \pi(w)$ for some $(F \cap F')$-timed automorphism π. We need to show

$$w \in L \quad \text{iff} \quad w' \in L,$$

which follows immediately by the following claim:

Claim 7.23.

Every $(F \cap F')$-timed automorphism π decomposes into $\pi = \pi_n \circ \cdots \circ \pi_1$, where each π_i is either an F- or an F'-timed automorphism.

Composition of timed automorphisms makes Π into a group. In short terms, claim 7.23 states that $\Pi_{F \cap F'} \subseteq \Pi_F + \Pi_{F'}$, where $\Pi_F + \Pi_{F'}$ is the smallest subgroup of Π including both Π_F and $\Pi_{F'}$. We state below in claim 7.25 a fact equivalent to claim 7.23, and which is based on the proof of Theorem 9.3 in [11]. An important ingredient of the proof of claim 7.25 is the following fact where, instead of dealing with decomposition of π, we analyse the individual orbit of $F \setminus F'$, in the special case when both $F \setminus F'$ and $F' \setminus F$ are singleton sets:

Claim 7.24.

Let $F, F' \subseteq [0, 1)$ be finite sets such that $F \setminus F' = \{\tau\}$ and $F' \setminus F = \{\tau'\}$. For every $(F \cap F')$-timed automorphism π we have $\pi(\tau) = (\pi_n \circ \cdots \circ \pi_1)(\tau)$, for some $\pi_1, \ldots, \pi_n$, each of which is either an F- or an F'-timed automorphism (i.e., belongs to $\Pi_F + \Pi_{F'}$).

Proof of claim 7.24.

We split the proof into two cases.

Case $F \cap F' \neq \varnothing$

Let κ be the greatest element of $F \cap F'$ smaller than τ, and let ϑ be the smallest element of $F \cap F'$ greater than τ, assuming they both exist. (If κ does not exist put $\kappa := \vartheta' - 1$, where ϑ' is the greatest element of $F \cap F'$; symmetrically, if ϑ does not exists put $\vartheta := \kappa' + 1$, where κ' is the smallest element of $F \cap F'$.) Then the $(F \cap F')$-orbit $\{\pi(\tau) \mid \pi \text{ is an } (F \cap F')\text{-timed automorphism}\}$ is the open interval (κ, ϑ). Take any $(F \cap F')$-timed automorphism π; without loss of generality assume that $\xi = \pi(\tau) > \tau$. The only interesting case is $\tau < \tau' \leq \xi$. In this case, we show $\pi(\tau) = \pi_2(\pi_1(\tau))$,where

- π_1 is some F'-timed automorphism that acts as the identity on $[\tau', \kappa + 1]$ and such that $\tau < \pi_1(\tau) < \tau'$,

- π_2 is some F-timed automorphism that acts as the identity on $[\vartheta - 1, \tau]$ and such that $\pi_2(\pi_1(\tau)) = \xi$.

Case $F \cap F' = \varnothing$

Thus $F = \{\tau\}$ and $F' = \{\tau'\}$. Take any timed automorphism π; without loss of generality assume that $\pi(\tau) > \tau$. Let $z \in \mathbb{Z}$ be the unique integer such that $\tau' + z - 1 < \tau < \tau' + z$. Let π_1 be an arbitrary $\{\tau'\}$-timed automorphism that maps τ to some $\tau_1 \in (\tau, \tau' + z)$. Note that τ_1 may be any value in $(\tau, \tau' + z)$. Similarly, let π_2 be an arbitrary $\{\tau\}$-timed automorphism that maps τ_1 to some $\tau_2 \in (\tau', \tau + 1)$. Again, τ_2 may be any value in $(\tau', \tau + 1)$. By repeating this process sufficiently many times one finally reaches $\pi(\tau)$ as required. $\square$

Claim 7.25.

Let $F, F' \subseteq [0, 1)$ be finite sets and let $G \subseteq \Pi$ be a subgroup of Π. If $\Pi_F \subseteq G$ and $\Pi_{F'} \subseteq G$ then $\Pi_{F \cap F'} \subseteq G$.

Proof of claim 7.25.

The proof is by induction on the size of the (finite) set $F \cup F'$. If $F \subseteq F'$ or $F' \subseteq F$, then the conclusion follows trivially. Otherwise, consider any $\tau \in F \setminus F'$ and $\tau' \in F' \setminus F$; obviously $\tau \neq \tau'$. Define $E = (F \cup F') \setminus \{\tau, \tau'\}$. We have $F \subseteq E \cup \{\tau\}$ and $F' \subseteq E \cup \{\tau'\}$ hence

$$\Pi_{E \cup \{\tau\}} \subseteq \Pi_F \subseteq G \qquad \Pi_{E \cup \{\tau'\}} \subseteq \Pi_{F'} \subseteq G.$$

We shall now prove that $\Pi_E \subseteq G$. To this end, consider any $\pi \in \Pi_E$. By claim 7.24, there exists a permutation

$$\psi = \pi_n \circ \cdots \circ \pi_1 \in \Pi_{E \cup \{\tau\}} + \Pi_{E \cup \{\tau'\}}$$

such that $\pi(\tau) = \psi(\tau)$. In other words, each of $\pi_1, \ldots, \pi_n$ is either a $(E \cup \{\tau\})$- or a $(E \cup \{\tau'\})$-timed automorphism. Since $\Pi_{E \cup \{\tau\}} \subseteq G$ and $\Pi_{E \cup \{\tau'\}} \subseteq G$, all $\pi_i \in G$, hence also $\psi \in G$.

On the other hand, clearly $\Pi_{E \cup \{\tau\}} \subseteq \Pi_E$ and $\Pi_{E \cup \{\tau'\}} \subseteq \Pi_E$, so all $\pi_i \in \Pi_E$, therefore $\psi \in \Pi_E$. As a result, $\pi^{-1} \circ \psi \in \Pi_E$. Since $(\pi^{-1} \circ \psi)(\tau) = \tau$, we obtain $\pi^{-1} \circ \psi \in \Pi_{E \cup \{\tau\}}$, therefore $\pi^{-1} \circ \psi \in G$. Together with $\psi \in G$ proved above, this gives $\pi \in G$. Thus we have proved $\Pi_E \subseteq G$.

It is now easy to show that $\Pi_{F \cap F'} \subseteq G$. Indeed, $|F \cup E| = |F \cup F'| - 1$, so by the inductive assumption for F and E, we have $\Pi_{F \setminus \{\tau\}} \subseteq G$ (note that $F \setminus \{\tau\} = F \cap E$). Further, $|(F \setminus \{\tau\}) \cup F'| = |F \cup F'| - 1$, so $\Pi_{F \cap F'} \subseteq G$ (note that $(F \setminus \{\tau\}) \cap F' = F \cap F'$), as required. $\square$

Claim 7.25 immediately implies claim 7.23 by taking $G = \Pi_F + \Pi_{F'}$. Lemma 7.22 is thus proved. $\square$

As a direct corollary of lemma 7.22, we have:

Corollary 7.26.

For every timed language L, the set $\{\mathsf{fract}(S) \mid S \subseteq_{\text{fin}} \mathbb{Q}_{\geq 0}, L \text{ is } S\text{-invariant}\}$, if nonempty, has a least (inclusion-wise) element.

Finally, recall S-orbits and orbits of elements, as defined abstractly in §2.3. Every bounded region corresponds to an orbit of configurations. Hence, in case of greedily resetting NTA where all reachable regions are bounded, orbits of reachable configurations are in bijective correspondence with reachable regions:

Fact 7.27.

Assume $\mathcal{A}$ is a greedily resetting $\mathsf{NTA}_{k,m}$. Two reachable configurations (p, μ, τ_0) and (p, μ', τ_0') of $\mathcal{A}$ with the same control location p have the same orbit if, and only if, the corresponding clock valuations $\tau_0 - \mu$ and $\tau_0' - \mu'$ belong to the same k, m-region.

7.3 Decidability of DTA_1 membership for NTA_1

In this section we prove our main decidability results for timed automata, which we now recall.

Theorem 7.28. DTA_k membership

For every fixed $k \in \mathbb{N}$, the DTA_k membership problem is decidable for NTA_1 languages.

Theorem 7.29. $\mathsf{DTA}_{k,m}$ membership

For every fixed $k, m \in \mathbb{N}$, the $\mathsf{DTA}_{k,m}$ membership problem is decidable for NTA_1 languages.

Both theorems are proved by means of the following key characterisation of those NTA_1 languages which are also DTA_k languages.

Lemma 7.30.

Let $\mathcal{A}$ be an $\mathsf{NTA}_{1,m}$ with n control locations, and let $k \in \mathbb{N}$.
The following conditions are equivalent:

1. $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$ for some always resetting DTA_k $\mathcal{B}$.
2. For every timed word w, there is $S \subseteq \mathbb{Q}_{\geq 0}$ of size at most k such that the last timestamp of w is in S and $w^{-1}\mathcal{L}(\mathcal{A})$ is S-invariant.
3. $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$ for some always resetting $\mathsf{DTA}_{k,m}$ $\mathcal{B}$ with at most $f(k, m, n) = \mathsf{Reg}(k, m) \cdot 2^{n(2km+1)}$ control locations ($\mathsf{Reg}(k, m)$ stands for the number of k, m-regions).

As in case of register automata, this characterisation provides a bound on the number of control locations of a DTA$_k$ equivalent to a given NTA$_1$ (if any exists).

The proof of theorem 7.28 builds on lemma 7.30 and on the following fact:

Lemma 7.31.

The DTA$_k$ and DTA$_{k,m}$ membership problems are both decidable for DTA languages.

Proof of lemma 7.31.

We reduce to a deterministic separability problem. Recall that a language S *separates* two languages L, M if $L \subseteq S$ and $S \cap M = \varnothing$.

It has recently been shown that the DTA$_k$ and DTA$_{k,m}$ separability problems are decidable for NTA [28, Theorem 1.1], and thus, in particular, for DTA.

To solve the membership problem, given a DTA $\mathcal{A}$, the procedure computes a DTA $\mathcal{A}'$ recognising the complement of $\mathcal{L}(\mathcal{A})$ and checks whether $\mathcal{A}$ and $\mathcal{A}'$ are DTA$_k$ separable (resp., DTA$_{k,m}$ separable) by using the result above.

It is a simple set-theoretic observation that $\mathcal{L}(\mathcal{A})$ is a DTA$_k$ language if, and only if, the languages $\mathcal{L}(\mathcal{A})$ and $\mathcal{L}(\mathcal{A}')$ are separated by some DTA$_k$ language, and likewise for DTA$_{k,m}$ languages. □

Proof of theorems 7.28 and 7.29.

We solve both problems in essentially the same way.

Given an NTA$_{1,m}$ $\mathcal{A}$, the decision procedure enumerates all always resetting DTA$_{k+1,m}$ $\mathcal{B}$ with at most $f(k+1, m, n)$ locations and checks whether $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$ (which is decidable by [79, Theorem 17]).

If no such DTA$_{k+1}$ $\mathcal{B}$ is found, the $\mathcal{L}(\mathcal{A})$ is not an always resetting DTA$_{k+1}$ language, due to lemma 7.30, and hence is not a DTA$_k$ language either; the procedure therefore answers negatively. Otherwise, in case when such a DTA$_{k+1}$ $\mathcal{B}$ is found, then a DTA$_k$ membership (resp. DTA$_{k,m}$ membership) test is performed on $\mathcal{B}$, which is decidable due to lemma 7.31. □

Remark 7.32. complexity

The decision procedure for NTA$_1$ invokes the HyperAckermann subroutine of [79] to check equivalence between an NTA$_1$ and a candidate DTA. This is in a sense unavoidable, since we show in theorem 7.48 that DTA$_k$ and DTA$_{k,m}$ membership problems are HyperAckermann-hard for NTA$_1$.

In the rest of this section we present the proof of lemma 7.30. The proof is a suitable extension and refinement of the argument used in case of register automata in §7.1.

7.3.1 Proof of lemma 7.30

Let us fix an $\mathsf{NTA}_{1,m}$ $\mathcal{A} = (\{\mathtt{x}\}, \Sigma, L, L_{\mathsf{ini}}, L_{\mathsf{fin}}, \Delta)$, where m is the greatest constant used in clock constraints in $\mathcal{A}$, and $k \in \mathbb{N}$. We assume w.l.o.g. that $\mathcal{A}$ is greedily resetting: This is achieved by resetting the clock as soon as upon reading an input symbol its value becomes greater than m or is an integer $\leq m$; we can record in the control location the actual integral value if it is $\leq m$, or a special flag otherwise. Consequently, after every discrete transition the value of the clock is at most m, and if it is an integer then it equals 0.

The implication (3) $\implies$ (1) follows by definition. For the implication (1) $\implies$ (2) suppose, by assumption, $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$ for a total always resetting DTA_k $\mathcal{B}$. Every left quotient $w^{-1}\mathcal{L}(\mathcal{A})$ equals $\mathcal{L}(\mathcal{B}; c)$ for some configuration c, hence point (2) follows by fact 7.21. Here we use the fact that $\mathcal{B}$ is always resetting in order to apply the second part of fact 7.21; without the assumption, we would only have S-invariance for sets S of size at most $k + 1$.

It thus remains to prove the implication (2) $\implies$ (3), which will be the content of the rest of the section. Assuming point (2), we are going to define an always resetting $\mathsf{DTA}_{k,m}$ $\mathcal{B}'$ with clocks $\mathtt{X} = \{\mathtt{x}_1, \dots, \mathtt{x}_k\}$ and with at most $f(k, m, n)$ locations such that $\mathcal{L}(\mathcal{B}') = \mathcal{L}(\mathcal{A})$. We start from the timed transition system $\mathcal{X}$ obtained by the finite powerset construction underlying the determinisation of $\mathcal{A}$, and then transform this transition system gradually, while preserving its language, until it finally becomes isomorphic to the reachable part of $[\![\mathcal{B}']\!]$ for some $\mathsf{DTA}_{k,m}$ $\mathcal{B}'$. As the last step we extract from this deterministic timed transition system a syntactic definition of $\mathcal{B}'$ and prove equality of their languages. This is achieved thanks to the invariance properties of the transition systems in the course of the transformation.

Macro-configurations

Configurations of the NTA_1 $\mathcal{A}$ are of the form $c = (p, \xi, \tau_0)$ where $\xi, \tau_0 \in \mathbb{Q}_{\geq 0}$ and $\xi \leq \tau_0$. A *macro-configuration* is a (not necessarily finite) set X of configurations (p, ξ, τ_0) of $\mathcal{A}$ which share the same value of the current timestamp τ_0, which we denote as $\mathsf{now}(X) = \tau_0$. We use the notation $\mathcal{L}(\mathcal{A}; X) := \bigcup_{c \in X} \mathcal{L}(\mathcal{A}; c)$. Let $\mathsf{succ}_{\sigma,\tau}(X) := \{c' \in [\![\mathcal{A}]\!] \mid c \xrightarrow{\sigma,\tau} c'$ for some $c \in X\}$ be the set of successors of configurations in X.

We define a deterministic timed transition system $\mathcal{X}$ consisting of the macro-configurations reachable in the course of determinisation of $\mathcal{A}$. Let $\mathcal{X}$ be the smallest set of macro-configurations and transitions such that

- $\mathcal{X}$ contains the initial macro-configuration: $X_0 = \{(p, 0, 0) \mid p \in L_{\mathsf{ini}}\} \in \mathcal{X}$;
- $\mathcal{X}$ is closed under successor: for every $X \in \mathcal{X}$ and $(\sigma, \tau) \in \Sigma \times \mathbb{Q}_{\geq 0}$, there is a transition $X \xrightarrow{\sigma,\tau} \mathsf{succ}_{a,\tau}(X)$ in $\mathcal{X}$.

Due to the fact that $[\![\mathcal{A}]\!]$ is finitely branching, i.e. $\mathsf{succ}_{\sigma,\tau}(\{c\})$ is finite for every fixed (σ, τ), all macro-configurations $X \in \mathcal{X}$ are finite. Let the final configurations of $\mathcal{X}$ be $F_{\mathcal{X}} = \{X \in \mathcal{X} \mid X \cap [\![\mathcal{A}]\!] \neq \varnothing\}$.

Claim 7.33.

$\mathcal{L}(\mathcal{A}; X) = \mathcal{L}(\mathcal{X}; X)$ for every $X \in \mathcal{X}$. In particular $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{X}; X_0)$.

For a macro-configuration X we write $\mathsf{Val}(X) := \{\xi \mid (p, \xi, \mathsf{now}(X)) \in X\} \cup \{\mathsf{now}(X)\}$ to denote the rationals appearing in X. Since $\mathcal{A}$ is greedily resetting, every macro-configuration $X \in \mathcal{X}$ satisfies $\mathsf{Val}(X) \subseteq (\mathsf{now}(X) - m, \mathsf{now}(X)]$. Whenever a macro-configuration X satisfies this condition we say that *the span of X is bounded by m*.

Pre-states

By assumption (Point 2), $\mathcal{L}(\mathcal{A}; X)$ is S-invariant for some S of size at most k, but the macro-configuration X itself needs not be S-invariant in general. Indeed, a finite macro-configuration $X \in \mathcal{X}$ is S-invariant if, and only if, $\mathsf{fract}(\mathsf{Val}(X)) \subseteq \mathsf{fract}(S)$, which is impossible in general when X is arbitrarily large, its span is bounded (by m), and size of S is bounded (by k). Intuitively, in order to assure S-invariance we will replace X by its S-closure $\mathsf{closure}_S(X)$ (recall fact 2.13).

A set $S \subseteq \mathbb{Q}_{\geq 0}$ is *fraction-independent* if it contains no two rationals with the same fractional part. A *pre-state* is a pair $Y = (X, S)$, where X is an S-invariant macro-state, and S is a finite fraction-independent subset of $\mathsf{Val}(X)$ that contains $\mathsf{now}(X)$. The intuitive rationale behind assuming the S-invariance of X is that it implies, together with the bounded span of X and the bounded size of S, that there are only finitely many pre-states, up to timed automorphism. We define the deterministic timed transition system $\mathcal{Y}$ as the smallest set of pre-states and transitions between them such that:

- $\mathcal{Y}$ contains the initial pre-state: $Y_0 = (X_0, \{0\}) \in \mathcal{Y}$;
- $\mathcal{Y}$ is closed under the closure of successor: for every $(X, S) \in \mathcal{Y}$ and $(\sigma, \tau) \in \Sigma \times \mathbb{Q}_{\geq 0}$, there is a transition $(X, S) \xrightarrow{\sigma,\tau} (X', S')$, where S' is the least, with respect to set inclusion, subset of $S \cup \{\tau\}$ containing τ such that the language $L' = (\sigma, \tau)^{-1}\mathcal{L}(\mathcal{A}; X) = \mathcal{L}(\mathcal{A}; \mathsf{succ}_{\sigma,\tau}(X))$ is S'-invariant, and $X' = \mathsf{closure}_{S'}(\mathsf{succ}_{\sigma,\tau}(X))$.

Observe that the least such fraction-independent subset S' exists due to the following facts: by fraction-independence of S there is a unique fraction-independent subset $\widetilde{S} \subseteq S \cup \{\tau\}$ which satisfies $\mathsf{fract}(\widetilde{S}) = \mathsf{fract}(S \cup \{\tau\})$ ($\widetilde{S}$ is obtained by removing from S any element ξ such that $\mathsf{fract}(\xi) = \mathsf{fract}(\tau)$, if any); since X is S-invariant, due to fact 7.20 so it is its language $\mathcal{L}(\mathcal{A}; X)$, and hence L' is necessarily $\widetilde{S}$-invariant; by assumption (Point 2), L' is R-invariant for some set $R \subseteq \mathbb{Q}_{\geq 0}$ of size at most k containing τ; let $\tau \subseteq [0, 1)$ be the least set of fractional values given by corollary 7.26 applied to L', i.e., $T \subseteq \mathsf{fract}(\widetilde{S}) \cap \mathsf{fract}(R)$; finally let $S' \subseteq \widetilde{S}$ be chosen so that $\mathsf{fract}(S') = T \cup \mathsf{fract}(\{\tau\})$. Due to fraction-independence of $\widetilde{S}$ the choice is unique and S' is fraction-independent. Furthermore, $\tau \in S'$ and the size of S' is at most k.

Example 7.34.

Suppose $k = 3$, $m = 2$, $\mathsf{succ}_{\sigma,\tau}(X) = \{(p, 3.7, 5), (q, 3.9, 5), (r, 4.2, 5)\}$ and $S' = \{3.7, 4.2, 5\}$. Then $X' = \{(p, 3.7, 5)\} \cup \{(q, \tau, 5) \mid \tau \in (3.7, 4.2)\} \cup \{(r, 4.2, 5)\}$. $\mathsf{now}(X') = 5$.

A corresponding *state* (as defined below) is (X', μ'), where $\mu' = \{\mathtt{x}_1 \mapsto 3.7, \mathtt{x}_2 \mapsto 4.2, \mathtt{x}_3 \mapsto 5\}$.

By fact 7.20, we deduce:

Claim 7.35. invariance of $\mathcal{Y}$

For every two transitions $(X_1, S_1) \xrightarrow{\sigma,\tau_1} (X_1', S_1')$ and $(X_2, S_2) \xrightarrow{\sigma,\tau_2} (X_2', S_2')$ in $\mathcal{Y}$ and a timed permutation π, if $\pi(X_1) = X_2$ and $\pi(S_1) = S_2$ and $\pi(\tau_1) = \tau_2$, then we have $\pi(X_1') = X_2'$ and $\pi(S_1') = S_2'$.

Proof of claim 7.35.

Let i range over $\{1, 2\}$ and let $\widetilde{X}_i \coloneqq \mathsf{succ}_{a,\tau_i}(X_i)$. Thus S_i' is the least subset of $S_i \cup \{\tau_i\}$ containing τ_i such that $\mathcal{L}(\mathcal{A}; \widetilde{X}_i)$ is S_i'-invariant, and $X_i' = \mathsf{closure}_{S_i'}(\widetilde{X}_i)$. By invariance of $[\![\mathcal{A}]\!]$ (fact 7.19) and invariance of semantics (fact 7.20) we get

$$\pi(\widetilde{X}_1) = \widetilde{X}_2, \qquad \text{and} \qquad \pi(\mathcal{L}(\mathcal{A}; \widetilde{X}_1)) = \mathcal{L}(\mathcal{A}; \widetilde{X}_2),$$

and therefore $\pi(S_1') = S_2'$, which implies $\pi(X_1') = X_2'$. □

Let the final configurations of $\mathcal{Y}$ be $F_{\mathcal{Y}} = \{(X, S) \in \mathcal{Y} \mid X \cap L_{\mathsf{fin}} \neq \varnothing\}$. By induction on the length of timed words it is easy to show:

Claim 7.36.

$\mathcal{L}(\mathcal{X}; X_0) = \mathcal{L}(\mathcal{Y}; Y_0)$.

Due to the assumption that $\mathcal{A}$ is greedily resetting and due to Point 2, in every pre-state $(X, S) \in \mathcal{Y}$ the span of X is bounded by m and the size of S is bounded by k.

States

We now introduce *states*, which are designed to be in one-to-one correspondence with configurations of the forthcoming DTA_k $\mathcal{B}'$. Intuitively, a state differs from a pre-state (X, S) only by allocating the values from S into k clocks, thus while a pre-state contains a set S, the corresponding state contains a reset-point assignment $\mu : \mathtt{X} \to \mathbb{Q}_{\geq 0}$ with image $\mu(\mathtt{X}) = S$.

Let $\mathtt{X} = \{\mathtt{x}_1, \ldots, \mathtt{x}_k\}$ be a set of k clocks. A *state* is a pair $Z = (X, \mu)$, where X is a macro-configuration, $\mu : \mathtt{X} \to \mathsf{Val}(X)$ is a reset-point assignment, $\mu(\mathtt{X})$ is a fraction-independent set containing $\mathsf{now}(X)$, and X is $\mu(\mathtt{X})$-invariant. Thus every state $Z = (X, \mu)$ determines uniquely a corresponding pre-state $\rho(Z) = (X, S)$ with $S = \mu(\mathtt{X})$. We define the deterministic timed transition system $\mathcal{Z}$ consisting of those states Z such that $\rho(Z) \in \mathcal{Y}$, and of transitions determined as follows: $(X, \mu) \xrightarrow{\sigma,\tau} (X', \mu')$ if the corresponding pre-state has a transition $(X, S) \xrightarrow{\sigma,\tau} (X', S')$ in $\mathcal{Y}$, where $S = \mu(\mathtt{X})$, and

$$\mu'(\mathtt{x}_i) \coloneqq \begin{cases} \tau & \text{if } \mu(\mathtt{x}_i) \notin S' \text{ or } \mu(\mathtt{x}_i) = \mu(\mathtt{x}_j) \text{ for some } j > i \\ \mu(\mathtt{x}_i) & \text{otherwise.} \end{cases} \tag{7.3}$$

Intuitively, the equation (7.3) defines a deterministic update of the reset-point assignment μ that amounts to resetting ($\mu'(\mathtt{x}_i) := \tau$) all clocks $\mathtt{x}_i$ whose value is either no longer needed (because $\mu(\mathtt{x}_i) \notin S'$), or is shared with some other clock x_j, for $j > i$ and is thus redundant. Due to this disciplined elimination of redundancy, knowing that $\tau \in S'$ and the size of S' is at most k, we ensure that at least one clock is reset in every step. In consequence, $\mu'(\mathtt{X}) = S'$, and the forthcoming DTA$_k$ $\mathcal{B}'$ will be always resetting. Using claim 7.35 we derive:

Claim 7.37. *invariance of $\mathcal{Z}$*

For every two transitions $(X_1, \mu_1) \xrightarrow{\sigma,\tau_1} (X_1', \mu_1')$ and $(X_2, \mu_2) \xrightarrow{\sigma,\tau_2} (X_2', \mu_2')$ in $\mathcal{Z}$ and a timed permutation π, if $\pi(X_1) = X_2$ and $\pi \circ \mu_1 = \mu_2$ and $\pi(\tau_1) = \tau_2$, then we have $\pi(X_1') = X_2'$ and $\pi \circ \mu_1' = \mu_2'$.

Proof of claim 7.37.

Let i range over $\{1, 2\}$. Let $S_i = \mu_i(\mathtt{X})$ and $(X_i, S_i) \xrightarrow{a,\tau_i} (X_i', S_i')$ in $\mathcal{Y}$. By claim 7.35 we have

$$\pi(X_1') = X_2' \qquad \text{and } \pi(S_1') = S_2'.$$

Since $\pi \circ \mu_1 = \mu_2$ and the definition (7.3) is invariant:

$$\pi \circ (\mu') = (\pi \circ \mu)',$$

we derive $\pi \circ \mu_1' = \mu_2'$. $\square$

Let the initial state be $Z_0 = (X_0, \mu_0)$, where $\mu_0(\mathtt{x}_i) = 0$ for all $\mathtt{x}_i \in \mathtt{X}$, and let the final states be $F_{\mathcal{Z}} = \{(X, \mu) \in \mathcal{Z} \mid X \cap [\![\mathcal{A}]\!] \neq \varnothing\}$. By induction on the length of timed words one proves:

Claim 7.38.

$\mathcal{L}(\mathcal{Y}; Y_0) = \mathcal{L}(\mathcal{Z}; Z_0)$.

In the sequel we restrict $\mathcal{Z}$ to states reachable from Z_0. In every state $Z = (X, \mu)$ in $\mathcal{Z}$, we have $\mathsf{now}(X) \in \mu(\mathtt{X})$. This will ensure the resulting DTA$_k$ $\mathcal{B}'$ to be always resetting.

Orbits of states

While a state is designed to correspond to a configuration of the forthcoming DTA$_k$ $\mathcal{B}'$, its orbit is designed to play the role of control location of $\mathcal{B}'$. We therefore need to prove that the set of states in $\mathcal{Z}$ is orbit-finite, i.e., the set of orbits $\{\mathsf{orbit}(Z) \mid Z \in \mathcal{Z}\}$ is finite and its size is bounded by $f(k, m, n)$. We start by deducing an analogue of fact 7.27:

Claim 7.39.

For two states $Z = (X, \mu)$ and $Z' = (X', \mu')$ in $\mathcal{Z}$, their reset-point assignments are in the same orbit, i.e., $\pi \circ \mu = \mu'$ for some $\pi \in \Pi$, if, and only if, the corresponding clock valuations $\mathsf{now}(X) - \mu$ and $\mathsf{now}(X') - \mu'$ belong to the same k, m-region.

(In passing note that, since in every state (X, μ) in $\mathcal{Z}$ the span of X is bounded by m, only bounded k, m-regions can appear in the last claim. Moreover, in each k, m-region one of the clocks constantly equals 0.) The action of timed automorphisms on macro-configurations and reset-point assignments is extended to states as $\pi(X, \mu) = (\pi(X), \pi\circ\mu)$. Recall that the orbit of a state Z is defined as $\mathsf{orbit}(Z) = \{\pi(Z) \mid \pi \in \Pi\}$.

Claim 7.40.

The number of orbits of states in $\mathcal{Z}$ is bounded by $f(k, m, n)$.

Proof of claim 7.40.

We finitely represent a state $Z = (X, \mu)$, relying on the following general fact.

Fact 7.41.

For every $\xi \in \mathbb{Q}_{\geq 0}$ and $S \subseteq \mathbb{Q}_{\geq 0}$, the S-orbit[2] $\mathsf{orbit}_S(\xi)$ is either the singleton $\{\xi\}$ (when $\xi \in S$) or an open interval with end-points of the form $\tau + z$ where $\tau \in S$ and $z \in \mathbb{Z}$ (when $\xi \notin S$).

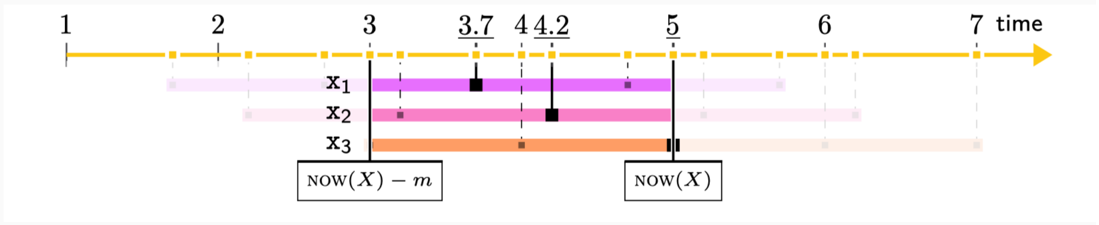
We apply the fact above to $S = \mu(\mathtt{X})$. In our case the span of X is bounded by m, and thus the same holds for $\mu(\mathtt{X})$.

Consequently, the integer z in the fact above always belongs to $\{-m, -m+1, \ldots, m\}$.

In turn, X splits into disjoint $\mu(\mathtt{X})$-orbits $\mathsf{orbit}_{\mu(\mathtt{X})}(\xi)$ consisting of open intervals separated by endpoints of the form $\tau + z$ where $\tau \in \mu(\mathtt{X})$ and $z \in \{-m, -m+1, \ldots, m\}$.

Example 7.42.

Continuing example 7.34, the endpoints are $\{3, 3.2, 3.7, 4, 4.2, 4.7, 5\}$, as shown in the illustration:



Recall that $\mu(\mathtt{X})$ is fraction-independent. Let $e_1 < e_2 < \cdots < e_{l+1}$ be all the endpoints of open-interval orbits ($l \leq km$), and let $o_1, o_2, o_3, \ldots := \{e_1\}, (e_1, e_2), \{e_2\}, \ldots$ be the consecutive S-orbits $\mathsf{orbit}_{\mu(\mathtt{X})}(\xi)$ of elements $\xi \in \mu(\mathtt{X})$. The number thereof is $2l + 1 \leq 2km + 1$. The finite representation of $Z = (X, \mu)$ consists of the pair (O, μ), where

$$O = \{(o_1, P_1), \ldots, (o_{2l+1}, P_{2l+1})\} \tag{7.4}$$

[2]The orbits of states Z should not be confused with S-orbits of individual rationals $\xi \in \mathbb{Q}_{\geq 0}$.

assigns to each orbit o_i the set of locations $P_i = \{p \,|\, (p, \xi, \tau_0) \in X$ for some $\xi \in o_i\} \subseteq L$, (which is the same as $P_i = \{p \,|\, (p, \xi, \tau_0) \in X$ for all $\xi \in o_i\}$ since X is $\mu(\mathtt{X})$-invariant, and hence $\mu(\mathtt{X})$-closed). Thus, a state $Z = (X, \mu)$ is uniquely determined by the sequence O as in (7.4) and the reset-point assignment μ.

We claim that the set of all the finite representations (O, μ), as defined above, is orbit-finite. Indeed, the orbit of (O, μ) is determined by the orbit of μ and the sequence

$$P_1,\ P_2,\ \ldots,\ P_{2km+1} \tag{7.5}$$

induced by the assignment O as in (7.4). Therefore, the number of orbits is bounded by the number of orbits of μ (which is bounded, due to claim 7.39, by $\mathsf{Reg}(k, m)$) times the number of different sequences of the form (7.5) (which is bounded by $(2^n)^{2km+1}$). This yields the required bound $f(k, m, n) = \mathsf{Reg}(k, m) \cdot 2^{n(2km+1)}$. □

Construction of the DTA

As the last step we define a DTA$_k$ $\mathcal{B}' = (\mathtt{X}, \Sigma, L', \{o_0\}, L'_{\mathsf{fin}}, \Delta')$ such that the reachable part of $[\![\mathcal{B}']\!]$ is isomorphic to $\mathcal{Z}$. Let locations $L' = \{\mathsf{orbit}(Z) \,|\, Z \in \mathcal{Z}\}$ be orbits of states from $\mathcal{Z}$, the initial location be the orbit o_0 of Z_0, and final locations $L'_{\mathsf{fin}} = \{\mathsf{orbit}(Z) \,|\, Z \in F_{\mathcal{Z}}\}$ be orbits of final states. A transition $Z = (X, \mu) \xrightarrow{\sigma,\tau} (X', \mu') = Z'$ in $\mathcal{Z}$ induces a transition rule in $\mathcal{B}'$

$$(o, a, \psi, \mathtt{Y}, o') \in \Delta' \tag{7.6}$$

whenever $o = \mathsf{orbit}(Z)$, $o' = \mathsf{orbit}(Z')$, ψ is the unique k, m-region satisfying $\tau - \mu \in [\![\psi]\!]$, and $\mathtt{Y} = \{\mathtt{x}_i \in \mathtt{X} \,|\, \mu'(\mathtt{x}_i) = \tau\}$. The automaton $\mathcal{B}'$ is indeed a DTA since o, σ and ψ uniquely determine $\mathtt{Y}$ and o':

Claim 7.43.

Suppose that two transitions $(X_1, \mu_1) \xrightarrow{\sigma,\tau_1} (X'_1, \mu'_1)$ and $(X_2, \mu_2) \xrightarrow{\sigma,\tau_2} (X'_2, \mu'_2)$ in $\mathcal{Z}$ induce transition rules $(o, \sigma, \psi, \mathtt{Y}_1, o'_1), (o, \sigma, \psi, \mathtt{Y}_2, o'_2) \in \Delta'$ with the same source location o and constraint ψ, i.e,

$$\tau_1 - \mu_1 \in [\![\psi]\!] \qquad \tau_2 - \mu_2 \in [\![\psi]\!]. \tag{7.7}$$

Then the target locations are equal $o'_1 = o'_2$, and the same for the reset sets $\mathtt{Y}_1 = \mathtt{Y}_2$.

(Notice that we only consider two transition rules with the same constraint ψ, instead of two different jointly satisfiable constraints ψ, ψ' as in the definition of deterministic timed automata, due to the fact that each constraint of $\mathcal{B}'$ is a single k, m-region.)

Proof of claim 7.43.

We use the invariance of semantics of $\mathcal{A}$ and claim 7.37. Let $o = \mathsf{orbit}(X_1, \mu_1) = \mathsf{orbit}(X_2, \mu_2)$. Thus there is a timed automorphism π such that

$$X_2 = \pi(X_1) \qquad \mu_2 = \pi \circ \mu_1. \tag{7.8}$$

It suffices to show that there is a (possibly different) timed permutation π' satisfying the following equalities:

$$\tau_2 = \pi'(\tau_1) \quad \{i \mid \mu'_1(\mathtt{x}_i) = \tau_1\} = \{i \mid \mu'_2(\mathtt{x}_i) = \tau_2\} \quad \mu'_2 = \pi' \circ \mu'_1 \quad X'_2 = \pi'(X'_1). \qquad (7.9)$$

We now rely the fact that both $\tau_{01} = \mathsf{now}(X_1) \in \mu_1(\mathtt{X})$ and $\tau_{02} = \mathsf{now}(X_2) \in \mu_2(\mathtt{X})$ are assigned to the same clock due to the second equality in (7.8): $\tau_{01} = \mu_1(\mathtt{x}_i)$ and $\tau_{02} = \mu_2(\mathtt{x}_i)$. We focus on the case when $\tau_1 - \tau_{01} \leq m$ (the other case is similar and easier since all clocks are reset due to greedy resetting), which implies $\tau_2 - \tau_{02} \leq m$ due to (7.7). In this case we may assume w.l.o.g., due to (7.7) and the equalities (7.8), that π is chosen so that $\pi(\tau_1) = \tau_2$. We thus take $\pi' = \pi$ for proving the equalities (7.9). Being done with the first equality, we observe that the last two equalities in (7.9) hold due to the invariance of $\mathcal{Z}$ (c.f. claim 7.37). The remaining second equality in (7.9) is a consequence of the third one. ☐

Claim 7.44.

Let $Z = (X, \mu)$ and $Z' = (X', \mu)$ be two states in $\mathcal{Z}$ with the same reset-point assignment. If $\pi(X) = X'$ and $\pi \circ \mu = \mu$ for some timed automorphism π then $X = X'$.

Claim 7.45.

$\mathcal{Z}$ is isomorphic to the reachable part of $[\![\mathcal{B}']\!]$.

Proof of claim 7.45.

We essentially repeat the argument of claim 7.15. For a state $Z = (X, \mu)$, let $\iota(Z) = (o, \mu, \tau)$, where $o = \mathsf{orbit}(Z)$ and $\tau = \mathsf{now}(X)$. Let $\mathcal{Z}'$ denote the reachable part of $[\![\mathcal{B}']\!]$. By claim 7.13, the mapping $\iota(\_)$ is a bijection between $\mathcal{Z}$ and its image $\iota(\mathcal{Z}) \subseteq [\![\mathcal{B}']\!]$. We aim at proving $\iota(\mathcal{Z}) = \mathcal{Z}'$. By the very definition (7.2), the image $\iota(\mathcal{Z})$ is a subsystem of $\mathcal{Z}'$. Recall that $\mathcal{Z}$ is total: for every $(\sigma_1, \tau_1) \dots (\sigma_n, \tau_n) \in \mathbb{T}(\Sigma)$, there is a sequence of transitions $(X_0, \mu_0) \xrightarrow{\sigma_1, \tau_1} \cdots \xrightarrow{\sigma_n, \tau_n}$ in $\mathcal{Z}$. Therefore $\iota(\mathcal{Z})$ is total too and, since $\mathcal{Z}'$ is deterministic and reachable, the subsystem $\iota(\mathcal{Z})$ necessarily equals $\mathcal{Z}'$. ☐

Claims 7.33, 7.36, 7.38 and 7.45 imply $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B}')$, which completes the proof of lemma 7.30.

7.4 Lower bounds for NTA

All the lower bounds in this section are obtained by a reduction from the universality problem for suitable language classes (does a given timed language $L \subseteq \mathbb{T}(\Sigma)$ satisfy $L = \mathbb{T}(\Sigma)$?), in complete analogy to §7.2 dealing with register automata. Our starting point is the following result.

Theorem 7.46. [46, theorem 1]

The DTA membership problem is undecidable for NTA languages.

We provide a suitable adaptation, generalization, and simplification of the result above which will allow us to extend undecidability to the DTA_k membership problem for every fixed $k \geq 0$, and also to obtain a complexity lower bound for NTA_1 input languages. Fix two languages $L \subseteq \mathbb{T}(\Sigma)$ and $M \subseteq \mathbb{T}(\Gamma)$, and a fresh alphabet symbol $\$ \not\in \Sigma \cup \Gamma$. The *composition* $L \triangleright M$ is the timed language over $\Sigma' = \Sigma \cup \{\$\} \cup \Gamma$ defined as follows:

$$L \triangleright M \;=\; \{u(\$, \tau)(v + \tau) \in \mathbb{T}(\Sigma') \mid u \in L, v \in M, \tau \in \mathbb{Q}_{\geq 0}\},$$

where τ is necessarily larger or equal than the last timestamp of u by the definition of $\mathbb{T}(\Sigma')$. The following lemma exposes some abstract conditions on classes of timed languages which are sufficient to encode the universality problem.

Lemma 7.47.

Let $k, m \in \mathbb{N}$ and let $\mathcal{Y}$ be a class of timed languages that

1. contains all the DTA_0 languages,
2. is closed under union and composition, and
3. contains some non-DTA_k (resp. non-$\mathsf{DTA}_{k,m}$) language.

The universality problem for languages in $\mathcal{Y}$ reduces in polynomial time to the DTA_k (resp. $\mathsf{DTA}_{k,m}$) membership problem for languages in $\mathcal{Y}$.

Lemma 7.47 is entirely analogous to lemma 7.16 for data languages, except that invariance of languages in $\mathcal{Y}$ is not required; moreover, notice that the notion of composition of timed languages that we need to state and prove the lemma above is a bit more complicated than the straightforward notion of concatenation that appears in the analogous statement for data languages from lemma 7.16.

Proof of lemma 7.47.

We consider DTA_k membership (the $\mathsf{DTA}_{k,m}$ membership is treated similarly). Consider some fixed timed language $M \in \mathcal{Y}$ which is not recognised by any DTA_k (relying on the assumption 3), over an alphabet Γ. For a given timed language $L \in \mathcal{Y}$, over an alphabet Σ, we construct the following language over the extended alphabet $\Sigma \cup \Gamma \cup \{\$\}$:

$$N \;\coloneqq\; L \triangleright \mathbb{T}(\Gamma) \;\cup\; \mathbb{T}(\Sigma) \triangleright M \;\subseteq\; \mathbb{T}(\Sigma \cup \Gamma \cup \{\$\}),$$

where $\$ \not\in \Sigma \cup \Gamma$ is a fixed fresh alphabet symbol. Since $\mathcal{Y}$ contains all the DTA_0 languages thanks to the assumption 1, and it is closed under union and composition thanks to the assumption 2, the language N belongs to $\mathcal{Y}$. We claim that the universality problem for L is equivalent to the DTA_k membership problem for N:

$$L = \mathbb{T}(\Sigma) \qquad \Longleftrightarrow \qquad N \text{ is recognised by a } \mathsf{DTA}_k.$$

For the "only if" direction, if $L = \mathbb{T}(\Sigma)$ then clearly $N = \mathbb{T}(\Sigma) \triangleright \mathbb{T}(\Gamma)$. Thus N is a DTA_0 languages, and thus also DTA_k for any $k \geq 0$. For the "if" direction suppose, towards reaching a contradiction, that N is recognised by a DTA_k $\mathcal{A}$ but $L \neq \mathbb{T}(\Sigma)$. Assume, w.l.o.g., that $\mathcal{A}$ is greedily resetting. Choose an arbitrary timed word $w =$

$(\sigma_1, \tau_1) \dots (\sigma_n, \tau_n) \notin L$ over Σ. Therefore, for any extension $v = (\sigma_1, \tau_1) \dots (\sigma_n, \tau_n)(\$, \tau_n + \tau)$ of w by one letter, we have

$$v^{-1}N = \tau + M = \{(\sigma'_1, \tau + \xi_1) \dots (\sigma'_m, \tau + \xi_m) \mid (\sigma'_1, \xi_1) \dots (\sigma'_m, \xi_m) \in M\}.$$

Choose τ larger than the largest absolute value m of constants appearing in clock constraints in $\mathcal{A}$, and let (p, μ) be the configuration reached by $\mathcal{A}$ after reading v. Since $\tau > m$, all the clocks are reset by the last transition and hence $\mu(\mathtt{x}) = 0$ for all clocks $\mathtt{x}$. Consequently, if the initial control location of $\mathcal{A}$ were moved to the location p, the so modified DTA_k $\mathcal{A}'$ would accept the language M. But this contradicts our initial assumption that M is not recognised by a DTA_k, thus finishing the proof. □

We can now prove the following refinement of theorem 7.46 claimed in the introduction.

Theorem 7.48. undecidability and hardness for DTA_k membership

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

1. undecidable for NTA_2,
2. undecidable for $\mathsf{NTA}_1^\varepsilon$ (NTA with epsilon transitions),
3. HyperAckermann-hard for NTA_1.

Proof of theorem 7.48.

Each of the three points follows by an application of lemma 7.47. For instance, for the first point take as $\mathcal{Y}$ the class of languages recognised by NTA_2. This class contains all DTA_0 languages, is closed under union and composition, and is not included in DTA_k for any k nor in $\mathsf{DTA}_{k,m}$ for any k, m (c.f. the NTA_1 language from example 2.24 which is not recognised by any DTA). Since the universality problem is undecidable for NTA_2 [2, Theorem 5.2], by lemma 7.47 the DTA_k and $\mathsf{DTA}_{k,m}$ membership problems are undecidable for NTA_2. The second and third points follow in the same way, using the fact that universality is undecidable for $\mathsf{NTA}_1^\varepsilon$ (NTA_1 with epsilon transitions) [70, Theorem 5.3], resp., HyperAckermann-hard for NTA_1 (by combining the same lower bound for the reachability problem in lossy channel systems [18, Theorem 5.5], together with the reduction from this problem to universality of NTA_1 given in [70, Theorem 4.1]). □

Chapter 8

Undecidability in case of unbounded number of registers/clocks

This chapter groups together four undecidability results that share a lot of similarities with one another. Problems are considered in the order of increasing complexity of proofs, latter building upon the former. Observe that all undecidable problems are the most general variants of the corresponding classes of problems—variants in which the number of registers/clocks of the sought deterministic automaton in not given in advance.

Theorem 8.1. DTA and $\mathsf{DTA}_{\bullet,m}$ membership und.

The DTA and $\mathsf{DTA}_{\bullet,m}$ $(m > 0)$ membership problems are undecidable for NTA_1 without epsilon transitions.

Theorem 8.2. synthesis of DTA und.

The timed synthesis decision problem is undecidable, and this holds already when Alice's winning condition is an NTA_2 language.

Theorem 8.3. DRA membership und.

The DRA membership problem is undecidable for NRA_1.

Theorem 8.4. synthesis of DRA und.

The register synthesis decision problem is undecidable, and this holds already when Alice's winning condition is an NRA_2 language.

In all cases, undecidability is shown by a reduction of the problem of *finiteness* for *lossy counter machines* (LCM) introduced in the §8.1. As a preparatory step, we introduce encodings of LCM runs as data and timed words. A core fact to be shown in the proofs concerning membership questions is that languages of incorrect encodings can be recognised by NRA/NTA. The last two sections, devoted to synthesis problems, additionally show that 'locally' correct encodings can be recognised as well.

8.1 Lossy counter machines

A *k-counter lossy counter machine* (LCM_k) is a tuple $\mathcal{M} = (\mathtt{C}, Q, q_{\mathsf{ini}}, \Delta)$, where $\mathtt{C}$ is a set of k counters, Q is a finite set of control locations, $q_{\mathsf{ini}} \in Q$ is the initial control location, and Δ is a finite set of instructions of the form $(p, \mathtt{op}, q)$, where $\mathtt{op} \in \mathtt{Op} = \mathtt{C} \times \{\text{'}\sqcup \mathtt{+= 1}\text{'}, \text{'}\sqcup \mathtt{-= 1}\text{'}, \text{'}\sqcup \mathtt{= 0?}\text{'}\}$. A configuration of an LCM $\mathcal{M}$ is a pair $(p, \mu) \in Q \times \mathbb{N}^{\mathtt{C}}$, where p is a control location, and μ is a counter valuation. For two counter valuations $\mu, \nu \in \mathbb{N}^{\mathtt{C}}$, we write $\mu \leq \nu$ if $\mu(\mathtt{c}) \leq \nu(\mathtt{c})$ for every counter $\mathtt{c} \in \mathtt{C}$. The semantics of an LCM $\mathcal{M}$ is given by a (potentially infinite) transition system over the configurations of $\mathcal{M}$ such that there is a transition $(p, \mu) \xrightarrow{\delta} (q, \nu)$, for $\delta = (p, \mathtt{op}, q) \in \Delta$, whenever one of the following conditions holds:

1. $\mathtt{op} = \text{'}\mathtt{c += 1}\text{'}$ and $\nu \leq \mu[\mathtt{c} \mapsto \mu(\mathtt{c}) + 1]$, or
2. $\mathtt{op} = \text{'}\mathtt{c -= 1}\text{'}$ and $\nu \leq \mu[\mathtt{c} \mapsto \mu(\mathtt{c}) - 1]$, or
3. $\mathtt{op} = \text{'}\mathtt{c = 0?}\text{'}$, $\nu(\mathtt{c}) = 0$[1] and $\nu \leq \mu$.

The *finiteness problem* (a.k.a. space boundedness) for an LCM $\mathcal{M}$ asks to decide whether the set of reachable configurations of $\mathcal{M}$

$$\mathsf{ReachConf}(\mathcal{M}) = \{(p, \mu) \mid (q_{\mathsf{ini}}, \mu_{\mathsf{ini}}) \to^* (p, \mu)\}$$

is finite, where $\mu_{\mathsf{ini}} = \lambda \mathtt{c}.0$ is the valuation mapping all counters to 0.

Theorem 8.5. [75, theorem 13]

The LCM_4 finiteness problem is undecidable.

8.2 Representing LCM runs as words

In this section, we devise an encoding of LCM runs as words over a finite alphabet, thus preparing grounds for the upcoming undecidability proofs. Our goal here is to provide a way of representing computations of LCM such that some of its properties are regular. In subsequent sections, the encoding will be further adapted to both data words and timed words. Languages of those extended encodings will turn out to be co-NRA and co-NTA languages, respectively.

Since already four lossy counters make the finiteness problem undecidable, for the rest of the chapter we will focus exclusively on LCM_4. Let us fix the LCM_4 $\mathcal{M} = (\mathtt{C}, Q, q_{\mathsf{ini}}, \Delta)$ with a set of counters $\mathtt{C} = \{\mathtt{c}_1, \mathtt{c}_2, \mathtt{c}_3, \mathtt{c}_4\}$. We start by providing encodings for counter valuations.

[1] Observe that the zero test can always be executed, as any counter can spontaneously lose its value due to the lossiness. The only requirement is that the corresponding counter value is 0 afterwards.

ENCODING A COUNTER VALUATION

Consider a counter valuation $\mu : \mathtt{C} \to \mathbb{N}$. Its *encoding* $\mathsf{enc}(\mu) \in \mathtt{C}^*$ is the word

$$\mathsf{enc}(\mu) = \mathtt{c}_1^{\mu(\mathtt{c}_1)}\,\mathtt{c}_2^{\mu(\mathtt{c}_2)}\,\mathtt{c}_3^{\mu(\mathtt{c}_3)}\,\mathtt{c}_4^{\mu(\mathtt{c}_4)}.$$

Let $\textsc{ValEnc}_{\mathcal{M}} = \mathcal{L}(\mathtt{c}_1^*\mathtt{c}_2^*\mathtt{c}_3^*\mathtt{c}_4^*)$ be the language of correct encodings of counter valuations. Additionally, for $w \in \textsc{ValEnc}_{\mathcal{M}}$, let $[\![w]\!]$ denote the unique valuation μ for which $\mathsf{enc}(\mu) = w$.

ENCODING LCM RUNS

For the reasons that will become apparent in next section, we will encode the LCM runs in a reverse order. To make the presentation consistent, we will also use a reversed notation $c' \xleftarrow{\delta} c$ for a transition $c \xrightarrow{\delta} c'$. Fix a run π of LCM

$$\pi = (p_1, \mu_1) \xleftarrow{\delta_1} (p_2, \mu_2) \xleftarrow{\delta_2} \cdots \xleftarrow{\delta_{n-2}} (p_{n-1}, \mu_{n-1}) \xleftarrow{\delta_{n-1}} (p_n, \mu_n) \tag{8.1}$$

starting in $(p_n, \mu_n) = (q_{\mathsf{ini}}, \mu_{\mathsf{ini}})$ and ending in (p_1, μ_1). Let $\Delta' = \Delta'$, and $\Sigma_{\mathcal{M}} = \mathtt{C} \cup \Delta'$. An *encoding* $\mathsf{enc}(\pi) \in \Sigma_{\mathcal{M}}^*$ of the run π is defined as:

$$\mathsf{enc}(\pi) = \triangleright\, \mathsf{enc}(\mu_1)\, \delta_1\, \mathsf{enc}(\mu_2)\, \delta_2 \cdots \delta_{n-2}\, \mathsf{enc}(\mu_{n-1})\, \delta_{n-1}\, \mathsf{enc}(\mu_n) \triangleleft \tag{8.2}$$

Note that $\mathsf{enc} : \mathsf{Runs}(\mathcal{M}) \hookrightarrow \Sigma_{\mathcal{M}}^*$. Let $\textsc{RunEnc}_{\mathcal{M}}$ be the language of encodings of runs of $\mathcal{M}$:

$$\textsc{RunEnc}_{\mathcal{M}} = \mathsf{enc}(\mathsf{Runs}(\mathcal{M})).$$

For $w \in \textsc{RunEnc}_{\mathcal{M}}$, by $[\![w]\!]$ we denote the unique run $\pi \in \mathsf{Runs}(\mathcal{M})$, such that $w = \mathsf{enc}(\pi)$.

When no additional assumptions on $\mathcal{M}$ are made, the language $\textsc{RunEnc}_{\mathcal{M}}$ needs not to be regular. However, many properties of valid encodings can be expressed as simple regular conditions. For this reason we define an auxiliary language $\textsc{Reg}_{\mathcal{M}} \supseteq \textsc{RunEnc}_{\mathcal{M}}$, which contains words $w \in \Sigma_{\mathcal{M}}^*$ satisfying conditions 8.6.A to 8.6.D, given below. As all those conditions can be easily verified by finite automata, the language is regular.

Intuitively, $\textsc{Reg}_{\mathcal{M}}$ accepts words which satisfy almost all properties of a valid encoding of LCM run. The only properties not captured by $\textsc{Reg}_{\mathcal{M}}$ are the inequalities given in points 1-3 in the definition of LCM, which are required to hold between consecutive counter valuations in a proper run's encoding.

Condition 8.6.A. **block structure**

Any valid encoding of an LCM run must have a block structure composed of alternating transition rules and counter valuation encodings:

$$w \in \triangleright\, \textsc{ValEnc}_{\mathcal{M}}\, (\Delta\, \textsc{ValEnc}_{\mathcal{M}})^* \triangleleft$$

Condition 8.6.B. **transition sequentiality**

In any run $\pi \in \mathsf{Runs}(\mathcal{M})$, successive transitions $p_i \xleftarrow{\delta_i} p_{i+1} \xleftarrow{\delta_{i+1}} p_{i+2}$ are labelled with transition rules sharing the intermediate state $p_{i+1} \in Q$:

$$\delta_{i+1} = (p_{i+2}, \mathsf{op}_{i+1}, \underline{p_{i+1}}), \qquad \delta_i = (\underline{p_{i+1}}, \mathsf{op}_i, p_i)$$

Therefore, in w, if we disregard symbols from $\Sigma_{\mathcal{M}} \setminus \Delta$, every two consecutive symbols $\delta_i, \delta_{i+1} \in \Delta$ need to be of the shape $(p, \_, \_)$ and $(\_, \_, p)$ for some $p \in Q$, respectively.

Observe that if w meets conditions 8.6.A and 8.6.B, it corresponds to an unique sequence $[\![w]\!]$ of LCM configurations interleaved with transition rules:

$$[\![w]\!] = (p_1, \mu_1) \xleftarrow{\delta_1} (p_2, \mu_2) \xleftarrow{\delta_2} \ldots \xleftarrow{\delta_{n-2}} (p_{n-1}, \mu_{n-1}) \xleftarrow{\delta_{n-1}} (p_n, \mu_n)$$

Condition 8.6.C. proper initial configuration

Every run $\pi \in \mathsf{Runs}(\mathcal{M})$ starts in the configuration $c_0 = (q_{\mathsf{ini}}, \lambda \mathtt{c}.0)$. Therefore, any proper encoding w must end (due to the reversed order) with the encoding of c_0 followed by $\triangleleft$:

$$w \in \Sigma^*_{\mathcal{M}} \, (q_{\mathsf{ini}}, \_, \_) \, \overbrace{\mathsf{enc}(\lambda \mathtt{c}.0)}^{\varepsilon} \, \triangleleft.$$

Condition 8.6.D. consistency with zero tests

The last regular requirement ensures that the zero tests appearing in w are reflected in corresponding counter valuations. For every infix of w of the form

$$\mathsf{enc}(\mu_i) \, (p_{i+1}, \text{‘}\mathtt{c} = 0?\text{’}, p_i) \, \mathsf{enc}(\mu_{i+1})$$

we require $\mathsf{enc}(\mu_i) \in (\mathtt{C} \setminus \{\mathtt{c}\})^*$, i.e., symbol $\mathtt{c}$ does not appear in $\mathsf{enc}(\mu_i)$.

8.3 Representing LCM runs as timed words

In this section, we define timed language $\textsc{RunEnc}^{\mathbb{T}}_{\mathcal{M}}$ tailored to satisfy, i.a., two design goals:

- $\mathsf{proj}_{\mathbb{T}}(\textsc{RunEnc}^{\mathbb{T}}_{\mathcal{M}}) = \textsc{RunEnc}_{\mathcal{M}}$,
- $\textsc{RunEnc}^{\mathbb{T}}_{\mathcal{M}} \in \text{co-NTA}_1$.

None of the above follows directly from the definition, the proof of the former concluding this section, and of the latter—being the main content of the next section.
Let $\textsc{Reg}^{\mathbb{T}}_{\mathcal{M}} = \mathsf{proj}^{-1}_{\mathbb{T}}(\textsc{Reg}_{\mathcal{M}})$. It clearly is in DTA_0. Define the timed encodings as

$$\textsc{RunEnc}^{\mathbb{T}}_{\mathcal{M}} = \textsc{Reg}^{\mathbb{T}}_{\mathcal{M}} \cap \mathsf{A}^{\mathbb{T}}_{\mathcal{M}} \cap \mathsf{B}^{\mathbb{T}}_{\mathcal{M}} \cap \mathsf{C}^{\mathbb{T}}_{\mathcal{M}}.$$

where $\mathsf{A}^{\mathbb{T}}_{\mathcal{M}}$, $\mathsf{B}^{\mathbb{T}}_{\mathcal{M}}$, and $\mathsf{C}^{\mathbb{T}}_{\mathcal{M}}$ be the languages of words satisfying conditions 8.7.A to 8.7.C below, respectively. For the sake of upcoming sections, we denote the complements (with respect to $\mathbb{T}(\Sigma_{\mathcal{M}})$) of languages arising in the definition as $\widehat{\textsc{Reg}}^{\mathbb{T}}_{\mathcal{M}}$, $\widehat{\mathsf{A}}^{\mathbb{T}}_{\mathcal{M}}$, $\widehat{\mathsf{B}}^{\mathbb{T}}_{\mathcal{M}}$, and $\widehat{\mathsf{C}}^{\mathbb{T}}_{\mathcal{M}}$.

Intuitively, the role of these conditions is to entail the property of LCM runs not captured by $\textsf{Reg}_{\mathcal{M}}$ —enforcing inequalities between successive counter valuations:

$$[\![v_{i+1}]\!](\mathsf{c}) \geq [\![v_i]\!](\mathsf{c}) + \begin{cases} 0 & \mathsf{op} \in \mathsf{Op} \setminus \{\text{`}c \mathrel{+}= 1\text{'}, \text{`}c \mathrel{-}= 1\text{'}\} \\ -1 & \mathsf{op} = \text{`}c \mathrel{+}= 1\text{'} \\ 1 & \mathsf{op} = \text{`}c \mathrel{-}= 1\text{'} \end{cases} \tag{8.3}$$

for every counter $\mathsf{c} \in \mathsf{C}$ and every two consecutive valuation encodings v_i, v_{i+1} with operation op executed between them. Note that in case of case '$c = 0?$', the actual test is made by $\textsf{Reg}_{\mathcal{M}}$).

Condition 8.7.A. strict monotonicity

Timed word w is strictly monotonic—it does not contain two letters with the same timestamp.

Condition 8.7.B. blocks aligned to unit intervals

Symbols from Δ' appear in w with consecutive integer timestamps starting from 0.

Observe that words belonging to $\textsf{Reg}^{\mathbb{T}}_{\mathcal{M}}$ and satisfying condition 8.7.B take a form

$$(\triangleright, 0)\ v_1\ (\delta_1, 1)\ v_2\ (\delta_2, 2) \cdots (\delta_{n-2}, n-2)\ v_{n-1}\ (\delta_{n-1}, n-1)\ v_n\ (\triangleleft, n) \tag{8.4}$$

for some $n \in \mathbb{N}$, where words v_i encode valuations $\mu_i = [\![\mathsf{proj}_{\mathbb{T}}(v_i)]\!]$.

Condition 8.7.C. well-alignment of valuation encodings

For every infix of w having the form $(\delta_{i-1}, \_)\ v_i\ ((p_{i+1}, \mathsf{op}, p_i), \_)\ v_{i+1}\ (\delta_{i+1}, \_)$, where $\delta_\bullet \in \Delta'$ and $v_\bullet \in \mathsf{proj}^{-1}_{\mathbb{T}}(\textsf{ValEnc}_{\mathcal{M}})$, the encodings v_i, v_{i+1} are *well aligned*, i.e., for each counter $\mathsf{c} \in \mathsf{C}$, depending on op, appropriate requirement is met:

Case a: $\mathsf{op} \in \mathsf{Op} \setminus \{\text{`}\mathsf{c} \mathrel{+}= 1\text{'}, \text{`}\mathsf{c} \mathrel{-}= 1\text{'}\}$, i.e., op does not modify the c counter

expected condition	*structural requirement*
$[\![v_{i+1}]\!](\mathsf{c}) \geq [\![v_i]\!](\mathsf{c})$	Each timed symbol (c, τ) in v_i implies existence of $(\mathsf{c}, \tau{+}1)$ in v_{i+1} after exactly one time unit.

Case b: $\mathsf{op} = \text{`}\mathsf{c} \mathrel{+}= 1\text{'}$

expected condition	*structural requirement*
$[\![v_{i+1}]\!](\mathsf{c}) \geq [\![v_i]\!](\mathsf{c}) - 1$	As in case a, with the exception of the first symbol (c, τ) in v_i, which can, but does not have to have a match in v_{i+1}.

Case c: $\mathsf{op} = \text{`}\mathsf{c} \mathrel{-}= 1\text{'}$

expected condition	*structural requirement*
$[\![v_{i+1}]\!](\mathsf{c}) \geq [\![v_i]\!](\mathsf{c}) + 1$	As in case a, and additionally in v_{i+1} an extra letter c appears within less than one time unit from the first symbol c of v_i.

We will show that $\textsf{RunEnc}^{\mathbb{T}}_{\mathcal{M}}$ characterises valid encodings of $\mathcal{M}$'s runs:

Lemma 8.8. timed encodings express all runs of $\mathcal{M}$, and only them

$\textsc{RunEnc}_{\mathcal{M}} = \mathsf{proj}_{\mathbb{T}}(\textsc{RunEnc}^{\mathbb{T}}_{\mathcal{M}})$.

Proof of lemma 8.8.

$\textsc{RunEnc}_{\mathcal{M}} \supseteq \mathsf{proj}_{\mathbb{T}}(\textsc{RunEnc}^{\mathbb{T}}_{\mathcal{M}})$

Fix $w \in \textsc{RunEnc}^{\mathbb{T}}_{\mathcal{M}}$. As $w \in \textsc{Reg}^{\mathbb{T}}_{\mathcal{M}}$, we know that w corresponds to a sequence

$$[\![w]\!] = \quad (p_1, \mu_1) \xleftarrow{\delta_1} (p_2, \mu_2) \xleftarrow{\delta_2} \dots \xleftarrow{\delta_{n-2}} (p_{n-1}, \mu_{n-1}) \xleftarrow{\delta_{n-1}} (p_n, \mu_n)$$

which is consistent with zero tests, the successive transition rules match, and it starts in a valid configuration. It remains to show that well-alignment implies that correct inequalities hold between every two consecutive valuations μ_i, μ_{i+1}. Fix arbitrary infix of w of the form

$$(\delta_{i-1}, \tau_{i-1})\; v_i\; ((p_{i+1}, \mathtt{op}, p_i), \tau_i)\; v_{i+1}\; (\delta_{i+1}, \tau_{i+1}).$$

Condition 8.7.B implies that the timestamps $\tau_\bullet$ are $i-1, i$ and $i+1$, respectively:

$$(\delta_{i-1}, i-1)\; v_i\; ((p_{i+1}, \mathtt{op}, p_i), i)\; v_{i+1}\; (\delta_{i+1}, i+1).$$

For any counter $\mathtt{c} \in \mathtt{C}$, if it is not modified by $\mathtt{op}$, well-alignment implies that every symbol $(\mathtt{c}, \tau)$ in v_i (with timestamp $\tau \in (i-1, i)$) has its repetition $(\mathtt{c}, \tau+1)$ in v_{i+1}. Due to strict monotonicity, the repetition $(\mathtt{c}, \tau+1)$ is not associated to any other symbols of v_i. Thus, $\mu_i(\mathtt{c}) \le \mu_{i+1}(\mathtt{c})$, as it was required. Other inequalities, for $\mathtt{op} \in \{\text{‘}\mathtt{c} \mathrel{+}= 1\text{’}, \text{‘}\mathtt{c} \mathrel{-}= 1\text{’}\}$ are handled similarly. Therefore, $[\![w]\!] \in \mathsf{Runs}(\mathcal{M})$.

$\textsc{RunEnc}_{\mathcal{M}} \subseteq \mathsf{proj}_{\mathbb{T}}(\textsc{RunEnc}^{\mathbb{T}}_{\mathcal{M}})$

We only need to show that $\mathtt{enc}(\pi)$, having form as in eq. (8.2), can be upgraded into a timed word satisfying conditions 8.7.A to 8.7.C. As there is only one, straightforward way of enforcing first two of them, it suffices to show that encodings of counter valuations can be well aligned. We construct the alignment from left to right, starting by arbitrarily selecting timestamps for $v_1 = \mathtt{enc}(\mu_1)$: we assign its i-th letter with $^{i+1}/_{|v_1|+2}$. Let us assume $v_i = \mathtt{enc}(\mu_i)$ has been already given timestamps and we need to align v_{i+1} well. Fix a counter $\mathtt{c} \in \mathtt{C}$. If $\mathtt{op}_i$ does not modify $\mathtt{c}$, we know that $\mu_i(\mathtt{c}) \le \mu_{i+1}(\mathtt{c})$, thus we may place one letter $\mathtt{c}$ one time unit later than every $\mathtt{c}$ appearing in v_i. The excessive letters $\mathtt{c}$ can be then distributed arbitrarily, e.g., after the last $\mathtt{c}$, but before the next symbol. Valuations of counters affected by presently executed operation can be aligned similarly. $\square$

8.4 Undecidability of DTA membership

This section contains a proof of the following undecidability result:

Theorem 8.1 — DTA and $\mathsf{DTA}_{\bullet,m}$ membership und.

The DTA and $\mathsf{DTA}_{\bullet,m}$ $(m > 0)$ membership problems are undecidable for NTA_1 without epsilon transitions.

The reduction presented here has two main ingredients—the lemmas stated below.

Lemma 8.9. — wrong timed encodings are recognised by NTA_1

$\mathbb{T}(\Sigma_{\mathcal{M}}) \setminus \textsc{RunEnc}^{\mathbb{T}}_{\mathcal{M}} \in \mathsf{NTA}_1$ and the automaton can be effectively constructed.

Lemma 8.10. — $\mathcal{M}$'s finiteness equivalent to determinisability of $\textsc{RunEnc}^{\mathbb{T}}_{\mathcal{M}}$

The set of reachable configurations $\mathsf{ReachConf}(\mathcal{M})$ is finite if, and only if, $\textsc{RunEnc}^{\mathbb{T}}_{\mathcal{M}}$ is a deterministic timed language.

Proof of theorem 8.1.

The reduction from theorem 8.5 is a result of a trivial juxtaposition of lemmas 8.9 and 8.10: given LCM_4 $\mathcal{M}$, construct the automaton for $\mathbb{T}(\Sigma_{\mathcal{M}}) \setminus \textsc{RunEnc}^{\mathbb{T}}_{\mathcal{M}}$ and ask about its determinisability. The answer will be positive if, and only if, the complement—$\textsc{RunEnc}^{\mathbb{T}}_{\mathcal{M}}$—is a deterministic timed language, which in turn occurs if, and only if, $\mathsf{ReachConf}(\mathcal{M})$ is finite. □

Proof of lemma 8.9.

Observe that

$$\mathbb{T}(\Sigma_{\mathcal{M}}) \setminus \textsc{RunEnc}^{\mathbb{T}}_{\mathcal{M}} = \widehat{\textsc{Reg}}^{\mathbb{T}}_{\mathcal{M}} \cup \widehat{\mathsf{A}}^{\mathbb{T}}_{\mathcal{M}} \cup \widehat{\mathsf{B}}^{\mathbb{T}}_{\mathcal{M}} \cup \widehat{\mathsf{C}}^{\mathbb{T}}_{\mathcal{M}}.$$

Since the class of NTA_1 is effectively closed under unions, we only need to show that languages $\widehat{\mathsf{A}}^{\mathbb{T}}_{\mathcal{M}}$, $\widehat{\mathsf{B}}^{\mathbb{T}}_{\mathcal{M}}$, and $\widehat{\mathsf{C}}^{\mathbb{T}}_{\mathcal{M}}$ are recognised by NTA_1. Thus, the statement to be proven stems from the following lemmas 8.11 to 8.13. □

Lemma 8.11.

$\widehat{\mathsf{A}}^{\mathbb{T}}_{\mathcal{M}} \in \mathsf{DTA}_1$.

Lemma 8.12.

$\widehat{\mathsf{B}}^{\mathbb{T}}_{\mathcal{M}} \in \mathsf{DTA}_1$.

Lemma 8.13.

$\widehat{\mathsf{C}}^{\mathbb{T}}_{\mathcal{M}} \in \mathsf{NTA}_1$.

Proof of lemma 8.11.

Observe that $w \in \widehat{\mathsf{A}}^{\mathbb{T}}_{\mathcal{M}}$ whenever two symbols in w appear with the same timestamp. Thus, $\widehat{\mathsf{A}}^{\mathbb{T}}_{\mathcal{M}}$ is straightforwardly recognised by DTA_1 with one clock $\mathtt{x}$ and just 3 states $L = \{\textsc{Ini}, \textsc{Ok}, \textsc{Err}\}$, first one being initial, $\textsc{Err}$ —accepting. The automaton has transitions

$$(\textsc{Ini} \xRightarrow{\sigma,\{\mathtt{x}\}} \textsc{Ok}) \qquad (\textsc{Ok} \xRightarrow{\sigma,\mathtt{x}>0,\{\mathtt{x}\}} \textsc{Ok}) \qquad (\textsc{Ok} \xRightarrow{\sigma,\mathtt{x}=0} \textsc{Err}) \qquad (\textsc{Err} \xRightarrow{\sigma} \textsc{Err}). \qquad \square$$

for each $\sigma \in \Sigma_{\mathcal{M}}$.

Proof of lemma 8.12.

Language $\widehat{\mathsf{B}}^{\mathbb{T}}_{\mathcal{M}}$ can be recognised by a DTA with one clock $\mathtt{x}$ as well. The corresponding automaton detects any misaligned boundary of blocks. As before, its states are $\{\textsc{Ini}, \textsc{Ok}, \textsc{Err}\}$, $\textsc{Ini}$ being initial and $\textsc{Err}$—accepting. The transitions are

$$(\textsc{Ini} \xRightarrow{\delta,\, \mathtt{x}=0,\, \{\mathtt{x}\}} \textsc{Ok}) \qquad (\textsc{Ok} \xRightarrow{\mathtt{c},\, \mathtt{x}<1} \textsc{Ok}) \qquad (\textsc{Ok} \xRightarrow{\delta,\, \mathtt{x}=1,\, \{\mathtt{x}\}} \textsc{Ok})$$

$$(\textsc{Ok} \xRightarrow{\sigma,\, \mathtt{x}>1} \textsc{Err}) \qquad (\textsc{Ok} \xRightarrow{\delta,\, \mathtt{x}<1} \textsc{Err}) \qquad (\textsc{Ok} \xRightarrow{\mathtt{c},\, \mathtt{x}=1} \textsc{Err}) \qquad (\textsc{Err} \xRightarrow{\sigma} \textsc{Err})$$

missing boundary — boundary too soon — bad boundary symbol

where $\sigma \in \Sigma_{\mathcal{M}}$, $\delta \in \Delta'$, and $\mathtt{c} \in \mathtt{C}$. □

Proof of lemma 8.13.

Language of counter valuation encodings' alignment errors, $\widehat{\mathsf{C}}^{\mathbb{T}}_{\mathcal{M}}$, is recognised by $\mathcal{E} \in \mathsf{NTA}_1$. Its definition is preceded by an intuitive description of its operation.

Intuition

The automaton tries to nondeterministically locate an infix w' of the form

$$w' = (\delta_{i-1}, \_)\, v_i\, ((p_{i+1}, \mathtt{op}, p_i), \_)\, v'_{i+1} \qquad \text{where } \delta_{i-1} \in \Delta' \text{ and } v_i, v'_{i+1} \in \mathbb{T}(\mathtt{C})$$

which witnesses a misalignment error, i.e., an infix such that for its every w'' of the form

$$w'' = v''_{i+1}\, (\delta_{i+1}, \_) \qquad \text{where } v'' \in \mathbb{T}(\mathtt{C}) \text{ and } \delta_{i+1} \in \Delta$$

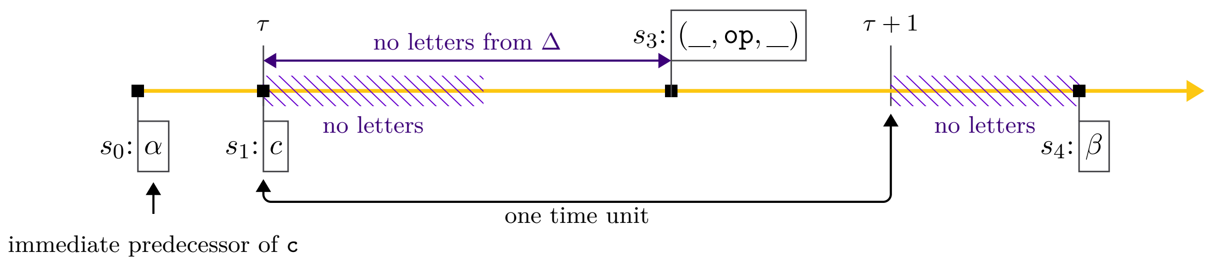
the word $w = w'w''$ does not satisfy the well-alignment condition 8.7.C. In general, for every $\mathtt{c} \in \mathtt{C}$, the condition can be violated in two ways:

1. symbol $s_1 = (\mathtt{c}, \tau)$ occurring in v_i is not followed by $(\mathtt{c}, \tau + 1)$ in v_{i+1}, while it should be, according to well-alignment, or
2. $\mathtt{op}$ = '$\mathtt{c}$ -= 1', a symbol $s_1 = (\mathtt{c}, \tau)$ is the first occurrence of $\mathtt{c}$ in v_i, but the last symbol with timestamp $< \tau + 1$ is not $\mathtt{c}$.

The automaton nondeterministically selects s_1 in v_i and tries to verify that one of the above cases occurs. To do so, it needs to locate the following symbols, if they exist, and store them in its control location:

- $s_0 = (\alpha, \_)$—the immediate predecessor of s_1,
- s_1—as above,
- $s_2 = ((\_, \mathtt{op}, \_), \_)$—the first timed symbol from Δ appearing after s_1,
- $s_3 = (\beta, \_)$—the last timed symbol in w with a timestamp $< \tau + 1$,
- $s_4 = (\gamma, \tau')$—the first timed symbol in w with $\tau' \geq \tau + 1$ (s_3 being its immediate predecessor)

All symbols are required to appear in the input word exactly in the order specified by their subscripts. For convenience, their mutual relations were illustrated in the picture below:



Having found the symbols, it checks whether one of the following conditions holds:

C1. op is '$\mathsf{c} \mathrel{-}= 1$' or it does not involve c, but $s_4 \neq (\mathsf{c}, \tau + 1)$,

C2. op = '$\mathsf{c} \mathrel{+}= 1$', s_1 is not the last letter in a sequence of symbols c (i.e., $\alpha = \mathsf{c}$), but $s_4 \neq (\mathsf{c}, \tau + 1)$,

C3. op = '$\mathsf{c} \mathrel{-}= 1$' and s_1 is the last letter in a sequence of symbols c ($\alpha \neq \mathsf{c}$), but the successor of s_4 is not labelled with c.

Upon successful verification, it goes to (and stays in) the accepting state ERR.

CONSTRUCTION OF THE AUTOMATON FOR $\widehat{\mathsf{C}}^{\mathbb{T}}_{\mathcal{M}}$

The state space of DTA$_1$ $\mathcal{E}$ realising the above is $\Sigma^0_{\mathcal{M}} \cup \Sigma^1_{\mathcal{M}} \cup \Sigma^2_{\mathcal{M}} \cup \Sigma^3_{\mathcal{M}} \cup \Sigma^4_{\mathcal{M}} \cup \{\text{ERR}\}$, the first component being used to find and store symbols $s_\bullet$. Its initial state is $()$, while ERR is the only accepting one. The automaton operates in three phases:

1. (finding s_1) In the first phase, it nondeterministically skips some prefix of the input and selects some letter $\mathsf{c} \in \mathsf{C}$, storing it by transitioning to state (s_1). At the same time, it resets its clock to be able to correctly determine s_4 later.

2. (finding s_2 and s_3) In the phase between guessing s_1 and finding s_3, automaton $\mathcal{E}$ checks for each symbol that the clock value is strictly less then 1—otherwise the run is not continued. Symbol $s_2 = (\alpha, \_)$—successor of s_1—is stored by going to state (c, α), if $\alpha \notin \Delta$, or to $(\mathsf{c}, \alpha, \alpha)$ otherwise. If letter from Δ is yet to be seen, i.e., the present state is (c, α), the automaton $\mathcal{E}$ waits with a self-loop and upon finding $\delta \in \Delta$ goes to $(\mathsf{c}, \alpha, \delta)$.

3. (finding s_4 and verifying conditions) In the last phase, being in state $(\mathsf{c}, \alpha, \delta)$, automaton $\mathcal{E}$ waits for s_4 with a self-loop requiring counter to be less than 1. Once it encounters symbol s_4 it can check conditions C1 and C2. If either one is met, it goes to (and stays in) state ERR. Otherwise, it goes to state $(\mathsf{c}, \alpha, \delta, \beta)$, and if the next letter read satisfies C3, it goes to ERR, too. □

Language of local errors $\widehat{\mathsf{C}}{\downarrow}^{\mathbb{T}}_{\mathcal{M}}$

For the sake of the upcoming §8.5 we also define an automaton $\mathcal{E}'$, which for every $\sigma \in \Sigma_{\mathcal{M}}$ does not contain the transition rule $(\textsc{Err} \overset{\sigma}{\Rightarrow} \textsc{Err})$. It recognises timed words, for which some run of $\mathcal{E}$ enters the $\textsc{Err}$ state for the first time. We denote the language of $\mathcal{E}'$ as $\widehat{\mathsf{C}}{\downarrow}^{\mathbb{T}}_{\mathcal{M}}$, and its complement as $\mathsf{C}{\downarrow}^{\mathbb{T}}_{\mathcal{M}}$. Intuitively, $\mathsf{C}{\downarrow}^{\mathbb{T}}_{\mathcal{M}}$ recognises a local absence of errors against the condition 8.7.C—no such error is detected whilst the last letter of the input is read.

Fact 8.14. properties of $\widehat{\mathsf{C}}{\downarrow}^{\mathbb{T}}_{\mathcal{M}}$

- $\widehat{\mathsf{C}}^{\mathbb{T}}_{\mathcal{M}} = \widehat{\mathsf{C}}{\downarrow}^{\mathbb{T}}_{\mathcal{M}} \cdot \mathbb{T}(\Sigma_{\mathcal{M}})$,
- $w \in \mathsf{C}^{\mathbb{T}}_{\mathcal{M}}$ if, and only if, its every prefix $v \sqsubseteq w$ belongs to $\mathsf{C}{\downarrow}^{\mathbb{T}}_{\mathcal{M}}$.

We move to the last remaining part of this section—the proof of lemma 8.10

Proof of lemma 8.10.

Given $\mathcal{M} \in \textsc{lcm}_4$, we need to prove the equivalence

$$|\mathsf{ReachConf}(\mathcal{M})| < \infty \iff \textsc{RunEnc}^{\mathbb{T}}_{\mathcal{M}} \in \textsc{dta}.$$

The '$\Rightarrow$' implication

Assume that $\mathsf{ReachConf}(\mathcal{M})$ is finite. Since the size of $\mathcal{M}$'s configuration space is limited, it becomes a finite state system and thus the language $\textsc{RunEnc}_{\mathcal{M}}$ is regular. However, to deterministically recognise $\textsc{RunEnc}^{\mathbb{T}}_{\mathcal{M}}$, we also need to verify conditions 8.7.A to 8.7.C. Let $k \in \mathbb{N}$ be the finite upper bound on the attainable sum of $\mathcal{M}$'s counter values. By lemma 8.8, in every $w \in \textsc{RunEnc}^{\mathbb{T}}_{\mathcal{M}}$ the length of infixes from $\textsc{ValEnc}_{\mathcal{M}}$is bounded by k, too. We will show that $\textsc{RunEnc}^{\mathbb{T}}_{\mathcal{M}}$ can be recognised by $\textsc{dta}_{2k+2}$[2]. Since $\mathsf{A}^{\mathbb{T}}_{\mathcal{M}}$ and $\mathsf{B}^{\mathbb{T}}_{\mathcal{M}}$ are complements of $\textsc{dta}_1$ languages (lemmas 8.11 and 8.12), they are $\textsc{dta}_1$.

We construct a timed automaton $\mathcal{A}$ which will recognise $\textsc{RunEnc}^{\mathbb{T}}_{\mathcal{M}}$. It simulates the automata recognising $\textsc{RunEnc}_{\mathcal{M}}$, $\mathsf{A}^{\mathbb{T}}_{\mathcal{M}}$, and $\mathsf{B}^{\mathbb{T}}_{\mathcal{M}}$. Therefore we can assume $[\![w]\!] \in \mathsf{Runs}(\mathcal{M})$ and w has the form as in eq. (8.4). Additionally, from the assumption, each v_i has length at most k.

Intuitively, we only need to deterministically verify condition 8.7.C (well-alignment) using remaining $2k$ clocks. To this end, we partition the clocks into two groups of size k, which will be used in alternating fashion, to verify alignment of pairs v_i, v_{i+1} starting in odd- and even-numbered blocks $v_\bullet$, respectively. Since well-alignment is a local property, the group of clocks can be reused after reading two blocks.

Fix pair of consecutive blocks v_i, v_{i+1}. When the automaton $\mathcal{A}$ reads v_i, it resets its j-th clock from the currently unused group and remembers each letter c_j of v_i in its state. Having read transition rule δ at the block boundary, $\mathcal{A}$ trivially verifies that the symbols from v_i are followed one time unit later by matching symbols in v_{i+1}, when well-alignment conditions require that.

[2] The construction is not optimal: $k+1$ clocks would suffice, coming at the price of a more involved proof.

The '$\Leftarrow$' implication

Let us assume that $\mathsf{ReachConf}(\mathcal{M})$ is infinite, and, towards contradiction, that $\textsc{RunEnc}^{\mathbb{T}}_{\mathcal{M}}$ is recognised by some $\mathcal{A} \in \text{DTA}$ with a finite set of clocks $\mathtt{X}$, $k = |\mathtt{X}|$. Due to first assumption, there is no finite upper bound on the attainable counter values. In particular, there exists a run π ending with a counter valuation μ with $\mu(\mathsf{c}) = k+2$. Due to , π has some encoding $w \in \textsc{RunEnc}^{\mathbb{T}}_{\mathcal{M}}$, which needs to have the form

$$w = \underbrace{\triangleright\ \overbrace{v'\ (\mathsf{c}, \tau_0) \ldots (\mathsf{c}, \tau_{k+1})\ v''}^{\mathsf{enc}(\mu)}\ (\delta, \tau)}_{w'}\ w'',$$

where v', v'' correspond to counters other than c, and $\delta \in \Delta$. When $\mathcal{A}$ is reading w, after observing the $\mathsf{enc}(\mu)$ block, it must forget at least two of the timestamps $\tau_0, \ldots, \tau_{k+1}$. At least one of them, say τ_j, is different than τ_0. Recall the well-alignment condition 8.7.C. Although depending on counter operation executed in between, symbol $(\mathsf{c}, \tau_0 + 1)$ may not be required in w'', it unconditionally needs to contain $k+1$ letters $(\mathsf{c}, \tau_1 + 1), \ldots, (\mathsf{c}, \tau_{k+1} + 1)$.

Intuitively, we will arrive at a contradiction by showing that the letter $(\mathsf{c}, \tau_j + 1)$ can be moved slightly without influencing the acceptance of $\mathcal{A}$. Let $z = (l, \nu)$ be the configuration of $\mathcal{A}$ after reading w'. Let $T = \nu(\mathtt{X} \cup \{\triangledown\})$ be the set of timestamps remembered in that configuration. Since $\mathcal{L}(z)$ is T-invariant (), and $\tau_j \notin T$, we may move the letter $(\tau_j + 1)$ slightly, by applying an automorphism $\psi \in \mathsf{Aut}_T(\mathbb{T})$ such that $\psi(\tau_j) \neq \tau_j$ to w''. The resulting encoding $w'\psi(w'')$ is no longer valid, since it violates condition 8.7.C, but $\mathcal{A}$ still accepts it. This contradicts that $\mathcal{A}$ recognises $\textsc{RunEnc}^{\mathbb{T}}_{\mathcal{M}}$, and thus $\textsc{RunEnc}^{\mathbb{T}}_{\mathcal{M}}$ is not a DRA language, as required. ☐

8.5 Undecidability of timed synthesis

In this section, we continue to develop the tools that helped to show the undecidability of the DTA membership problem, aiming this time to prove the decision version of timed synthesis problem undecidable:

Theorem 8.2 synthesis of DTA und.

The timed synthesis decision problem is undecidable, and this holds already when Alice's winning condition is an NTA_2 language.

For $\mathcal{M} \in \text{LCM}_4$, we will define a timed synthesis game $\mathsf{G}^{\mathbb{T}}_{\mathcal{M}}$ with condition $W_{\mathcal{M}}$ and show that:

Lemma 8.15. Alice's winning condition in $\mathsf{G}^{\mathbb{T}}_{\mathcal{M}}$ is NTA_2

$W_{\mathcal{M}} \in \text{NTA}_2$.

Lemma 8.16. $\mathsf{G}^{\mathbb{T}}_{\mathcal{M}}$ characterises determinisability of $\textsc{RunEnc}^{\mathbb{T}}_{\mathcal{M}}$

$\mathsf{ReachConf}(\mathcal{M})$ is finite if, and only if, there exists a winning strategy for Bob in $\mathsf{G}^{\mathbb{T}}_{\mathcal{M}}$.

Proof of theorem 8.2.

Undecidability is a trivial consequence of theorem 8.5 and lemmas 8.15 and 8.16. ☐

8.5.1 Definition of $\mathsf{G}^{\mathbb{T}}_{\mathcal{M}}$

Players in game $\mathsf{G}^{\mathbb{T}}_{\mathcal{M}}$ have actions

$$\begin{aligned} A &= \Sigma_{\mathcal{M}} && \text{(Alice)} \\ B &= \{\textsf{OK}, \rightsquigarrow, \textsf{ERRR}, \textsf{ERRA}, \textsf{ERRB}, \textsf{ERRC}\} && \text{(Bob)} \end{aligned}$$

The winning set of Alice is $W_{\mathcal{M}} = \textsc{Reach}(V^{\mathbb{T}}_{\mathcal{M}})$ for the following language $V^{\mathbb{T}}_{\mathcal{M}}$ of finite timed words:

$$\begin{aligned} V^{\mathbb{T}}_{\mathcal{M}} = {} & \mathsf{proj}_{\mathsf{B}}^{-1}(\textsf{OK}^* \rightsquigarrow^+ \textsf{OK}) && (8.5)\\ & \cup \mathsf{proj}_{\mathsf{B}}^{-1}(\textsf{OK}^* \rightsquigarrow^* \rightsquigarrow) \cap \mathsf{proj}_{\mathsf{A}}^{-1}(\textsc{Reg}^{\mathbb{T}}_{\mathcal{M}}) && (8.6)\\ & \cup \mathsf{proj}_{\mathsf{B}}^{-1}(\textsf{OK}^* \rightsquigarrow^* \textsf{ERRR}) \cap \mathsf{proj}_{\mathsf{A}}^{-1}(\mathsf{R}^{\mathbb{T}}_{\mathcal{M}}) && (8.7)\\ & \cup \mathsf{proj}_{\mathsf{B}}^{-1}(\textsf{OK}^* \rightsquigarrow^* \textsf{ERRA}) \cap \mathsf{proj}_{\mathsf{A}}^{-1}(\mathsf{A}^{\mathbb{T}}_{\mathcal{M}} \cup \widehat{\mathsf{R}}^{\mathbb{T}}_{\mathcal{M}}) && (8.8)\\ & \cup \mathsf{proj}_{\mathsf{B}}^{-1}(\textsf{OK}^* \rightsquigarrow^* \textsf{ERRB}) \cap \mathsf{proj}_{\mathsf{A}}^{-1}(\mathsf{B}^{\mathbb{T}}_{\mathcal{M}} \cup \widehat{\mathsf{R}}^{\mathbb{T}}_{\mathcal{M}} \cup \widehat{\mathsf{A}}^{\mathbb{T}}_{\mathcal{M}}) && (8.9)\\ & \cup \mathsf{proj}_{\mathsf{B}}^{-1}(\textsf{OK}^* \rightsquigarrow^* \textsf{ERRC}) \cap \mathsf{proj}_{\mathsf{A}}^{-1}(\mathsf{C}{\downarrow}^{\mathbb{T}}_{\mathcal{M}} \cup \widehat{\mathsf{R}}^{\mathbb{T}}_{\mathcal{M}} \cup \widehat{\mathsf{A}}^{\mathbb{T}}_{\mathcal{M}} \cup \widehat{\mathsf{B}}^{\mathbb{T}}_{\mathcal{M}}) && (8.10)\\ & \cup \mathsf{proj}_{\mathsf{B}}^{-1}(\textsf{OK}^* \textsf{OK}) \cap \mathsf{proj}_{\mathsf{A}}^{-1}(\widehat{\mathsf{R}}^{\mathbb{T}}_{\mathcal{M}} \cup \widehat{\mathsf{A}}^{\mathbb{T}}_{\mathcal{M}} \cup \widehat{\mathsf{B}}^{\mathbb{T}}_{\mathcal{M}} \cup \widehat{\mathsf{C}}{\downarrow}^{\mathbb{T}}_{\mathcal{M}}) && (8.11) \end{aligned}$$

8.5.2 Intuitition

Before we move to the proof of lemmas 8.15 and 8.16, we first provide a less formal, intuitive description of the mechanics of game $\mathsf{G}^{\mathbb{T}}_{\mathcal{M}}$.

Fix a play of $\mathsf{G}^{\mathbb{T}}_{\mathcal{M}}$ —an infinite timed word $w = (a_1, b_1, \tau_1)(a_2, b_2, \tau_2) \cdots \in \mathbb{T}^{\omega}(A \cdot B)$. Let v_n denote its prefix of length $n \in \mathbb{N}$. Conceptually, in n-th turn, Bob has provide an answer b_n to some question regarning the timed word $\mathsf{proj}_{\mathsf{A}}(v_n)$ built so far by Alice. Alice wins if Bob gives an incorrect answer in some round; conversely, Bob wins if all his answers are correct.

The question for Bob is: 'Does the word $\mathsf{proj}_{\mathsf{A}}(v_n)$ belong to one of the languages $\widehat{\mathsf{R}}^{\mathbb{T}}_{\mathcal{M}}$, $\widehat{\mathsf{A}}^{\mathbb{T}}_{\mathcal{M}}$, $\widehat{\mathsf{B}}^{\mathbb{T}}_{\mathcal{M}}$, $\widehat{\mathsf{C}}{\downarrow}^{\mathbb{T}}_{\mathcal{M}}$ of erroneous encodings? If so, point out the first out of the listed languages to which it belongs'.

Intuition on the Bob's actions

Bellow is a list of permitted responses to this question, along with their intuitive meanings:

$\textsf{OK}$	'$\mathsf{proj}_{\mathsf{A}}(v_n)$ does not belong to any of the languages of incorrect encodins',
$\textsf{ERR}\mathsf{R}$, $\textsf{ERR}\mathsf{A}$, $\textsf{ERR}\mathsf{B}$, $\textsf{ERR}\mathsf{C}$	'$\mathsf{proj}_{\mathsf{A}}(v_n)$ belongs to $\widehat{\mathsf{R}}^{\mathbb{T}}_{\mathcal{M}}$, $\widehat{\mathsf{A}}^{\mathbb{T}}_{\mathcal{M}}$, $\widehat{\mathsf{B}}^{\mathbb{T}}_{\mathcal{M}}$, or $\widehat{\mathsf{C}}^{\mathbb{T}}_{\mathcal{M}}$, respectively, but it does not belong to any language appearing earlier in the list',
$\rightsquigarrow$	'I cannot point out any of the required errors in $\mathsf{proj}_{\mathsf{A}}(v_n)$ just yet, but I am certain that as Alice continues to build the word, some error will appear no later than the end of encoding marker $\triangleleft$ is reached'.

Bob is punished for giving an incorrect answer by $V^{\mathbb{T}}_{\mathcal{M}}$.

Intuition on the components of $V^{\mathbb{T}}_{\mathcal{M}}$

Every line of $V^{\mathbb{T}}_{\mathcal{M}}$'s definition has the form $\mathsf{proj}_{\mathsf{B}}^{-1}(L_{\mathsf{B}}) \cap \mathsf{proj}_{\mathsf{A}}^{-1}(L_{\mathsf{A}})$ for some regular language L_{B} and timed language L_{A}. A prefix $v \sqsubseteq w$ is winning for Alice, if it satisfies both $\mathsf{proj}_{\mathsf{B}}(v) \in L_{\mathsf{B}}$ and $\mathsf{proj}_{\mathsf{A}}(v) \in L_{\mathsf{A}}$. Let us analyse the definition of $V^{\mathbb{T}}_{\mathcal{M}}$ line by line, intuitively describing the requirements they impose on Bob upon reading v.

(8.5)	Bob is forbidden to use $\textsf{OK}$ after he played $\rightsquigarrow$ which leaves him only with the remaining symbols $\{\rightsquigarrow, \textsf{ERR}\mathsf{R}, \textsf{ERR}\mathsf{A}, \textsf{ERR}\mathsf{B}, \textsf{ERR}\mathsf{C}\}$. Intuitively, by playing $\rightsquigarrow$, he declared he will find an error in the future and he cannot withdraw this promise.
(8.6)	Bob is forbidden to play $\rightsquigarrow$ when $v \in \mathsf{proj}_{\mathsf{A}}^{-1}(\textsf{REG}^{\mathbb{T}}_{\mathcal{M}})$ (i.e., when, i.a., the last symbol Alice produced is $\triangleleft$). Intuitively, a past use of $\rightsquigarrow$ obliges Bob to play a symbol from $\{\textsf{ERR}\mathsf{R}, \textsf{ERR}\mathsf{A}, \textsf{ERR}\mathsf{B}, \textsf{ERR}\mathsf{C}\}$ the turn Alice plays $\triangleleft$ at the latest.
(8.7) – (8.10)	Bob's error reports need to correctly point out the first language among $\widehat{\mathsf{R}}^{\mathbb{T}}_{\mathcal{M}}$, $\widehat{\mathsf{A}}^{\mathbb{T}}_{\mathcal{M}}$, $\widehat{\mathsf{B}}^{\mathbb{T}}_{\mathcal{M}}$, and $\widehat{\mathsf{C}}^{\mathbb{T}}_{\mathcal{M}}$ to which $\mathsf{proj}_{\mathsf{A}}(v)$ belongs. Bob loses both when he fails to notice an error with higher priority than the one being reported, or when his report is false.
(8.11)	Bob cannot play $\textsf{OK}$ when $\mathsf{proj}_{\mathsf{A}}(v)$ does belong to some language of erroneous encodings.

Observe that in each of (8.5) – (8.11) the $\mathsf{proj}_{\mathsf{B}}^{-1}(L_{\mathsf{B}})$ component features a language L_{B} with the property that a letter from $E = \{\textsf{ERR}\mathsf{R}, \textsf{ERR}\mathsf{A}, \textsf{ERR}\mathsf{B}, \textsf{ERR}\mathsf{C}\}$ appears at most once. Therefore, the outcome of the game is decided as soon as Bob plays some symbol from E.

Additionally, note that if a partial play v_n features the symbol $a_n = \triangleleft$, and Alice has not won yet (i.e., $v_i \notin V^{\mathbb{T}}_{\mathcal{M}}$ for $i \leq n$), from now on Bob has a trivial winning strategy: to reply with $\textsf{ERR}\mathsf{R}$ no matter what his opponent plays. Indeed, his answer will be correct, as no partial play $v \sqsupset v_n$ containing v_n as its proper prefix can satisfy $\mathsf{proj}_{\mathsf{A}}(v) \in \mathsf{R}^{\mathbb{T}}_{\mathcal{M}}$, because a prefix of $\textsf{REG}^{\mathbb{T}}_{\mathcal{M}}$ can only contain $\triangleleft$ as the last letter.

For a play $w \in \mathbb{T}^\omega(A \cdot B)$, let $\mathsf{cut}(w) \in \mathbb{T}(A \cdot B)$ denote its prefix up to and including the first symbol of the form $(\triangleleft, \_)$ if it exists, and an empty prefix ε otherwise. The winning condition is designed to ensure the following property:

Lemma 8.17. correspondence between $\textsc{RunEnc}^{\mathbb{T}}_{\mathcal{M}}$ and plays of $\mathsf{G}^{\mathbb{T}}_{\mathcal{M}}$

Consider a play $w \in \mathsf{Plays}(\mathsf{G}^{\mathbb{T}}_{\mathcal{M}}) \setminus W_{\mathcal{M}}$ with nonempty prefix $v = \mathsf{cut}(w)$. Then

$$\mathsf{proj}_{\mathsf{A}}(v) \in \textsc{RunEnc}^{\mathbb{T}}_{\mathcal{M}} \iff \mathsf{proj}_{\mathsf{B}}(v) \in \textsc{ok}^*.$$

In words, if w is winning for Bob and $\mathsf{cut}(w)$ is not empty, then $\mathsf{proj}_{\mathsf{A}}(v)$ is a valid encoding of $\mathcal{M}$'s run if, and only if, Bob played only $\textsc{ok}$ in v. This characterisation will play a crucial role in proving lemma 8.16.

8.5.3 Proofs

The remainder of this section contains proofs of lemmas 8.15 to 8.17.

Proof of lemma 8.15.

Observe that it suffices to prove that $V^{\mathbb{T}}_{\mathcal{M}} \in \mathsf{NTA}_2$. Since the class of NTA_2 is closed with respect to unions, we only need to show that each of the languages stated in lines (8.5) – (8.11) is NTA_2. This is evident for all lines but (8.10), because we have already shown that $\textsc{Reg}^{\mathbb{T}}_{\mathcal{M}}, \mathsf{R}^{\mathbb{T}}_{\mathcal{M}}, \mathsf{A}^{\mathbb{T}}_{\mathcal{M}}, \mathsf{B}^{\mathbb{T}}_{\mathcal{M}} \in \mathsf{DTA}_1$ and $\widehat{\mathsf{C}{\downarrow}}^{\mathbb{T}}_{\mathcal{M}} \in \mathsf{NTA}_1$. Thus, we only need to show that

$$\mathsf{proj}_{\mathsf{B}}^{-1}(\textsc{ok}^* \rightsquigarrow^* \textsc{err}\mathsf{C}) \cap \mathsf{proj}_{\mathsf{A}}^{-1}(\mathsf{C}{\downarrow}^{\mathbb{T}}_{\mathcal{M}} \cup \widehat{\mathsf{R}}^{\mathbb{T}}_{\mathcal{M}} \cup \widehat{\mathsf{A}}^{\mathbb{T}}_{\mathcal{M}} \cup \widehat{\mathsf{B}}^{\mathbb{T}}_{\mathcal{M}}) \in \mathsf{NTA}_2,$$

the difficulty being caused by the occurence of $\mathsf{C}{\downarrow}^{\mathbb{T}}_{\mathcal{M}}$. Since the component $\mathsf{proj}_{\mathsf{B}}^{-1}(\textsc{ok}^* \rightsquigarrow^* \textsc{err}\mathsf{C})$ is just a regular condition on the word played by Bob, we can focus on proving that

$$\mathsf{C}{\downarrow}^{\mathbb{T}}_{\mathcal{M}} \cup \widehat{\mathsf{R}}^{\mathbb{T}}_{\mathcal{M}} \cup \widehat{\mathsf{A}}^{\mathbb{T}}_{\mathcal{M}} \cup \widehat{\mathsf{B}}^{\mathbb{T}}_{\mathcal{M}} \in \mathsf{NTA}_2.$$

We construct $\mathcal{A} \in \mathsf{NTA}_2$ recognising that language. It consists of four independent components, thus recognising the union of their languages. The components are $\mathcal{A}'$ (yet to be defined), and the automata corresponding to languages $\widehat{\mathsf{R}}^{\mathbb{T}}_{\mathcal{M}}$, $\widehat{\mathsf{A}}^{\mathbb{T}}_{\mathcal{M}}$, and $\widehat{\mathsf{B}}^{\mathbb{T}}_{\mathcal{M}}$. This construction guarantees that $\widehat{\mathsf{R}}^{\mathbb{T}}_{\mathcal{M}}, \widehat{\mathsf{A}}^{\mathbb{T}}_{\mathcal{M}}, \widehat{\mathsf{B}}^{\mathbb{T}}_{\mathcal{M}} \subseteq \mathcal{L}(\mathcal{A})$. To complete the proof, we need to design $\mathcal{A}' \in \mathsf{NTA}_2$ such that for every $w \in \mathsf{R}^{\mathbb{T}}_{\mathcal{M}} \cap \mathsf{A}^{\mathbb{T}}_{\mathcal{M}} \cap \mathsf{B}^{\mathbb{T}}_{\mathcal{M}}$

$$w \in \mathcal{L}(\mathcal{A}') \Leftrightarrow w \in \mathsf{C}{\downarrow}^{\mathbb{T}}_{\mathcal{M}}.$$

This in turn boils down to verifying whether the automaton $\mathcal{E}'$ recognising $\widehat{\mathsf{C}{\downarrow}}^{\mathbb{T}}_{\mathcal{M}}$ from the fact 8.14 rejects w. Recall that in order to accept, $\mathcal{E}'$ tries to locate in w symbols s_1, s_2, s_3, s_4 satisfying particular conditions (C1, C2 or C3).

□

Proof of lemma 8.16.

The '⇒' implication

Assume that $\mathsf{ReachConf}(M)$ is finite. There is some k such that every reachable configuration (p, μ) has size $\sum_{\mathsf{c}\in\mathsf{C}} \mu(\mathsf{c}) \leq k$. In this case, the set of correct timed encodings of runs of M can be recognised by a $\mathsf{DTA}_{(k+2)}$ $\mathcal{A}$ which resets clock $\mathtt{x}_j$ when reading the j-th position of block $p_i u_i \delta_i$ (which is of length $\leq k+2$). From $\mathcal{A}$ we can produce a winning controller for Bob with k clocks:

The '⇐' implication

Assume that $\mathsf{ReachConf}(\mathcal{M})$ is infinite and, towards reaching a contradiction, that Bob has a winning controller $\mathcal{B} = (\mathtt{X}, A, B, L, l_{\mathsf{ini}}, \Delta)$ with $k \in \mathbb{N}$ clocks. We will focus on constructing $\mathcal{B}' \in \mathsf{DTA}_k$ recognising $\mathsf{RunEnc}^{\mathbb{T}}_{\mathcal{M}}$, which will imply by lemma 8.10 that $\mathsf{ReachConf}(\mathcal{M})$ is finite.

Construction of DTA$_k$

Let $\mathcal{B}' = (\mathtt{X}, A, L \cup \{\mathsf{Ok}, \mathsf{Err}\}, l_{\mathsf{ini}}, \{\mathsf{Ok}\}, \Delta')$, where Δ' contains transitions

- $(p \xRightarrow{a,\varphi,\mathtt{Y}} q)$ for every $(p \xRightarrow{a/\mathsf{ok},\varphi,\mathtt{Y}} q) \in \Delta$ where $a \in A \setminus \{\triangleleft\}$,
- $(p \xRightarrow{a,\varphi,\mathtt{Y}} \mathsf{Ok})$ for every $(p \xRightarrow{\triangleleft/\mathsf{ok},\varphi,\mathtt{Y}} q) \in \Delta$,
- $(p \xRightarrow{a,\varphi,\mathtt{Y}} \mathsf{Err})$ for every $(p \xRightarrow{a/e,\varphi,\mathtt{Y}} q) \in \Delta$, where $e \in B \setminus \{\mathsf{ok}\}$,
- $(\mathsf{Err} \xRightarrow{a} \mathsf{Err})$ for every $a \in A$.

Correctness

We need to show that $\mathcal{L}(\mathcal{B}') = \mathsf{RunEnc}^{\mathbb{T}}_{\mathcal{M}}$. Take any $u \in \mathbb{T}(A)$. Let w be any $(u, \mathcal{B})$-conformant play of $\mathsf{G}^{\mathbb{T}}_{\mathcal{M}}$. As $\mathcal{B}$ is a winning controller, w is winning for Bob. Let $v = \mathsf{cut}(w)$. Note that by construction $\mathsf{proj}_{\mathsf{A}}(v) = u$, and v corresponds to an accepting run of $\mathcal{B}'$ if, and only if, $\mathsf{proj}_{\mathsf{B}}(v) \in \mathsf{ok}^*$. Therefore we have

$$u \in \mathcal{L}(\mathcal{B}') \iff \mathsf{proj}_{\mathsf{B}}(v) \in \mathsf{ok}^* \iff \mathsf{proj}_{\mathsf{A}}(v) = u \in \mathsf{RunEnc}^{\mathbb{T}}_{\mathcal{M}}$$

where the last equivalence follows from lemma 8.17. □

Proof of lemma 8.17.

Let us fix a play $w \in \mathsf{Plays}(\mathsf{G}^{\mathbb{T}}_{\mathcal{M}}) \setminus W_{\mathcal{M}}$ winning for Bob, such that $v = \mathsf{cut}(w)$ is not empty. We need to show that

$$\mathsf{proj}_{\mathsf{A}}(v) \in \mathsf{RunEnc}^{\mathbb{T}}_{\mathcal{M}} \iff \mathsf{proj}_{\mathsf{B}}(v) \in \mathsf{ok}^*$$

The '⇒' implication

Assume $\mathsf{proj}_{\mathsf{A}}(v) \in \mathsf{RunEnc}^{\mathbb{T}}_{\mathcal{M}} = \mathsf{Reg}^{\mathbb{T}}_{\mathcal{M}} \cap \mathsf{A}^{\mathbb{T}}_{\mathcal{M}} \cap \mathsf{B}^{\mathbb{T}}_{\mathcal{M}} \cap \mathsf{C}^{\mathbb{T}}_{\mathcal{M}}$. Towards contradiction, assume that $\mathsf{proj}_{\mathsf{B}}(v) \notin \mathsf{ok}^*$. We aim at showing that w is not winning for Bob. Note that it suffuces to exhibit a prefix $v' \sqsubseteq v$ belonging to $V^{\mathbb{T}}_{\mathcal{M}}$. Let $E = \{\mathsf{errR}, \mathsf{errA}, \mathsf{errB}, \mathsf{errC}\}$.

CASE 1: $\mathsf{proj}_{\mathsf{B}}(v) \in B^*EB^*$

Let $v' \sqsubseteq v$ be the shortest prefix of v such that the last letter b of $\mathsf{proj}_{\mathsf{B}}(v')$ belongs to E. Since w is winning, $\mathsf{proj}_{\mathsf{B}}(v') \in \textsc{ok}^* \rightsquigarrow^* E$ (as $\textsc{ok}$ cannot follow $\rightsquigarrow$). Note that $\mathsf{proj}_{\mathsf{A}}(v') \in \mathsf{R}^{\mathbb{T}}_{\mathcal{M}} \cap \mathsf{A}^{\mathbb{T}}_{\mathcal{M}} \cap \mathsf{B}^{\mathbb{T}}_{\mathcal{M}} \cap \mathsf{C}{\downarrow}^{\mathbb{T}}_{\mathcal{M}}$, because

- ▶ $\mathsf{R}^{\mathbb{T}}_{\mathcal{M}}$ is the prefix closure of $\textsc{Reg}^{\mathbb{T}}_{\mathcal{M}}$,
- ▶ languages $\mathsf{A}^{\mathbb{T}}_{\mathcal{M}}$ and $\mathsf{B}^{\mathbb{T}}_{\mathcal{M}}$ are both prefix-closed, and
- ▶ $\mathsf{proj}_{\mathsf{A}}(v') \in \mathsf{C}{\downarrow}^{\mathbb{T}}_{\mathcal{M}}$ due to fact 8.14.

therefore, depending on the letter $b \in E$, one of the components (8.7) – (8.10) matches v', implying $v' \in V^{\mathbb{T}}_{\mathcal{M}}$.

CASE 2: $\mathsf{proj}_{\mathsf{B}}(v) \in \{\textsc{ok}, \rightsquigarrow\}^*$

In this case $\mathsf{proj}_{\mathsf{B}}(v)$ needs to contain symbol $\rightsquigarrow$. If it is followed by $\textsc{ok}$ symbol, some prefix of v trivially matches the component (8.5) of $v_{\mathcal{M}}$'s definition. Otherwise $\mathsf{proj}_{\mathsf{B}}(v) \in \textsc{ok}^*\rightsquigarrow^+$. But $\mathsf{proj}_{\mathsf{A}}(v) \in \textsc{Reg}^{\mathbb{T}}_{\mathcal{M}}$ and therefore $v \in V^{\mathbb{T}}_{\mathcal{M}}$ (cf. (8.6)).

The '$\Leftarrow$' implication

Assume that $\mathsf{proj}_{\mathsf{B}}(v) \in \textsc{ok}^*$. Since w is winning, no prefix of v belongs to $V^{\mathbb{T}}_{\mathcal{M}}$. Therefore, for every $v' \sqsubseteq v$ we have $\mathsf{proj}_{\mathsf{A}}(v') \in \mathsf{R}^{\mathbb{T}}_{\mathcal{M}} \cap \mathsf{A}^{\mathbb{T}}_{\mathcal{M}} \cap \mathsf{B}^{\mathbb{T}}_{\mathcal{M}} \cap \mathsf{C}{\downarrow}^{\mathbb{T}}_{\mathcal{M}}$ (cf. (8.11)). Using fact 8.14 we conclude that $\mathsf{proj}_{\mathsf{A}}(v) \in \mathsf{C}^{\mathbb{T}}_{\mathcal{M}}$, and—consequently—$\mathsf{proj}_{\mathsf{A}}(v) \in \textsc{RunEnc}^{\mathbb{T}}_{\mathcal{M}}$. □

8.6 Representing LCM runs as data words

In this section, we set up the general framework for the upcoming undecidability results related to register automata. Although their general idea and structure will remain similar to the timed case, there will be quite significant differences in the technical details. In particular, membership and game undecidability proofs diverge sooner, as they require slightly different encoding variants. Therefore, we define two languages of encodings: $\textsc{RunEnc}^{\mathbb{A}}_{\mathcal{M}}$ (unordered) and $\textsc{RunEnc}^{\mathbb{A},<}_{\mathcal{M}}$ (ordered). Analogously to §8.3, they will have the properties

- ▶ $\mathsf{proj}_{\mathbb{A}}(\textsc{RunEnc}^{\mathbb{A}}_{\mathcal{M}}) = \mathsf{proj}_{\mathbb{A}}(\textsc{RunEnc}^{\mathbb{A},<}_{\mathcal{M}}) = \textsc{RunEnc}_{\mathcal{M}}$ (cf. corollaries 8.22 and 8.23),
- ▶ $\textsc{RunEnc}^{\mathbb{A}}_{\mathcal{M}} \in \textsc{co-NRA}_1$,
- ▶ $\textsc{RunEnc}^{\mathbb{A},<}_{\mathcal{M}} \in \textsc{co-NRA}_2$.

We specify both languages indirectly, by imposing some of conditions 8.18.A to 8.18.C$^<$. Let $\textsc{Reg}^{\mathbb{A}}_{\mathcal{M}} = \mathsf{proj}^{-1}_{\mathbb{A}}(\textsc{Reg}_{\mathcal{M}})$. It clearly belongs to $\textsc{dra}_0$. Define

$$\textsc{RunEnc}^{\mathbb{A}}_{\mathcal{M}} = \textsc{Reg}^{\mathbb{A}}_{\mathcal{M}} \cap \mathsf{A}^{\mathbb{A}}_{\mathcal{M}} \cap \mathsf{B}^{\mathbb{A}}_{\mathcal{M}} \cap \mathsf{C}^{\mathbb{A}}_{\mathcal{M}}$$
$$\textsc{RunEnc}^{\mathbb{A},<}_{\mathcal{M}} = \textsc{Reg}^{\mathbb{A}}_{\mathcal{M}} \cap \mathsf{A}^{\mathbb{A}}_{\mathcal{M}} \cap \mathsf{B}^{\mathbb{A}}_{\mathcal{M}} \cap \mathsf{C}^{\mathbb{A},<}_{\mathcal{M}}$$

where $\mathsf{A}^{\mathbb{A}}_{\mathcal{M}}$, $\mathsf{B}^{\mathbb{A}}_{\mathcal{M}}$, $\mathsf{C}^{\mathbb{A}}_{\mathcal{M}}$, and $\mathsf{C}^{\mathbb{A},<}_{\mathcal{M}}$ are the languages of words satisfying conditions 8.18.A to 8.18.C$^<$ below, respectively. Again, the complements with respect to $(\Sigma_{\mathcal{M}} \times \mathbb{A})^*$ are denoted as $\widehat{\mathsf{REG}}^{\mathbb{A}}_{\mathcal{M}}$, $\widehat{\mathsf{A}}^{\mathbb{A}}_{\mathcal{M}}$, $\widehat{\mathsf{B}}^{\mathbb{A}}_{\mathcal{M}}$, $\widehat{\mathsf{C}}^{\mathbb{A}}_{\mathcal{M}}$, and $\widehat{\mathsf{C}}^{\mathbb{A},<}_{\mathcal{M}}$.

Similarly to the timed case, these conditions are built to entail inequalities between successive counter valuations (cf. eq. (8.3)). Recall that $w \in \mathsf{REG}^{\mathbb{A}}_{\mathcal{M}}$ implies w has a block structure imposed by condition 8.6.A.

Condition 8.18.A. **all transitions come with single, unique data value**

There exists $\alpha \in \mathbb{A}$ such that for all symbols $(\sigma, \beta) \in \Sigma_{\mathcal{M}} \times \mathbb{A}$ of the input word

$$\sigma \in \Delta' \iff \beta = \alpha.$$

Intuitively, all symbols from Δ' appear with α, and just them.

Condition 8.18.B. **no duplication of data values within blocks**

Every valuation encoding block in w has all its atoms pairwise different. (But the same atom can appear in many blocks.)

Condition 8.18.C. **symbols are repeated**

Choose any counter $\mathtt{c} \in \mathtt{C}$ and arbitrary suffix of w having the form

$$(\delta_0, \_)\ v_1\ (\delta_1, \alpha_1)\ v_2\ w' \tag{8.12}$$

where ▶ $\delta_\bullet \in \Delta'$, ▶ $\delta_1 = (p_2, \mathtt{op}, p_1)$, ▶ $\alpha_\bullet \in \mathbb{A}$, ▶ $\mathsf{proj}_{\mathbb{A}}(v_\bullet) \in \mathtt{C}^*$, and ▶ v_2 is a maximal infix containing only letters from $\mathtt{C}$. Depending on $\mathtt{op}$, we require the following:

Case a: $\mathtt{op} \in \mathtt{Op} \setminus \{\text{‘}c \mathrel{+}= 1\text{’}, \text{‘}c \mathrel{-}= 1\text{’}\}$, i.e., $\mathtt{op}$ does not modify $\mathtt{c}$

expected condition	*structural requirement*
$[\![v_2]\!](\mathtt{c}) \geq [\![v_1]\!](\mathtt{c})$	Each letter/atom pair $(\mathtt{c}, \alpha)$ in v_1 reappears in v_2.

Case b: $\mathtt{op} = \text{‘}\mathtt{c} \mathrel{+}= 1\text{’}$

expected condition	*structural requirement*
$[\![v_2]\!](\mathtt{c}) \geq [\![v_1]\!](\mathtt{c}) - 1$	As in case a, with the exception of the first symbol labeled with $\mathtt{c}$ in v_1, which can, but does not have to have a match in v_2.

Case c: $\mathtt{op} = \text{‘}\mathtt{c} \mathrel{-}= 1\text{’}$

expected condition	*structural requirement*
$[\![v_2]\!](\mathtt{c}) \geq [\![v_1]\!](\mathtt{c}) + 1$	As in case a, and additionally the repetition $(\mathtt{c}, \alpha)$ in v_2 cannot be the first occurrence of $\mathtt{c}$ in v_2.

The final condition 8.18.C$^<$ modifies slightly 8.18.C. Below, neighbouring atoms of v_1, if they are repeated in v_2, they need to be so in the same order as in v_1.

Condition 8.18.C$^<$. pairs of symbols, if repeated, appear in order

Choose any counter $\mathtt{c} \in \mathtt{C}$ and arbitrary suffix of w having the form

$$(\delta_0, \_)\ v_1\ (\delta_1, \alpha_1)\ v_2\ w' \tag{8.13}$$

where ▶ $\delta_\bullet \in \Delta'$, ▶ $\delta_1 = (p_2, \mathtt{op}, p_1)$, ▶ $\alpha_\bullet \in \mathbb{A}$, ▶ $\mathsf{proj}_{\mathbb{A}}(v_\bullet) \in \mathtt{C}^*$, and ▶ v_2 is a maximal infix containing letters from $\mathtt{C}$ in all positions except the last one, which can be arbitrary. Depending on $\mathtt{op}$, we require the following:

Case a': $\mathtt{op} \in \mathtt{Op} \setminus \{`c \mathrel{+}= 1\text{'}, `c \mathrel{-}= 1\text{'}\}$, i.e., $\mathtt{op}$ does not refer to $\mathtt{c}$

expected condition	*structural requirement*
$[\![v_2]\!](\mathtt{c}) \geq [\![v_1]\!](\mathtt{c})$	For any two consecutive symbols $xx' = (\mathtt{c}, \beta)\ (\_, \beta')$ in $v_1\ (\delta_1, \alpha_1)$, if atom β' appears in v_2, then x occurs in there too, before β' does.

Cases b-c are an exact copy of cases 8.18.C.b-c, but now defined on top of updated condition a'.

Observe that $\mathsf{C}^{\mathbb{A},<}_{\mathcal{M}} \not\subseteq \mathsf{C}^{\mathbb{A}}_{\mathcal{M}}$, because condition 8.18.C$^<$ is also met by $w = (\delta_1, \_)\ v\ (\delta_2, \alpha_2)\ (\delta_3, \alpha_3)$, for arbitrary $v \in (\mathtt{C} \times \mathbb{A})^*$ and atom α_3 different from atoms of $v\ (\delta_2, \alpha_2)$. However, once we enforce that all symbols from Δ' are labeled with one atom $\alpha \in \mathbb{A}$, we obtain

Lemma 8.19. conditions 8.18.A and 8.18.C$^<$ together imply condition 8.18.C

$\mathsf{A}^{\mathbb{A}}_{\mathcal{M}} \cap \mathsf{C}^{\mathbb{A},<}_{\mathcal{M}} \subseteq \mathsf{C}^{\mathbb{A}}_{\mathcal{M}}$.

Proof of lemma 8.19.

Fix a data word $u \in (\Sigma_{\mathcal{M}} \times \mathbb{A})^*$. Assume that it meets conditions 8.18.A and 8.18.C$^<$. Take arbitrary infix w of u having the form as in eq. (8.12). Let us enumerate symbols of v_1, starting from the last: $v_1 = x_n \ldots x_2 x_1 x_0$, where $n \in \mathbb{N}$ and $x_i = (\mathtt{c}_i, \beta_i) \in \mathtt{C} \times \mathbb{A}$. The proof is a simple induction with respect to i.

Base case

Observe that condition 8.18.A implies that $\alpha_1 = \alpha_2$. Applying condition 8.18.C$^<$ to w, choosing pair of symbols $x_0(\delta, \alpha_1)$ as xx', we get that x_0 appears in v_2.

Inductive step

This follows trivially from the inductive assumption that x_i reappears in v_2, and condition 8.18.C$^<$, applied to the pair $x_{i+1}x_i$. □

Lemma 8.19 trivially implies that

Corollary 8.20. ordered encodings are valid unordered encodings

$\textsc{RunEnc}^{\mathbb{A},<}_{\mathcal{M}} \subseteq \textsc{RunEnc}^{\mathbb{A}}_{\mathcal{M}}$.

The remainder of this section is devoted to the proofs of the following lemma:

Lemma 8.21.

- $\textsc{RunEnc}_{\mathcal{M}} \subseteq \mathsf{proj}_{\mathbb{A}}(\textsc{RunEnc}^{\mathbb{A},<}_{\mathcal{M}}) \subseteq \mathsf{proj}_{\mathbb{A}}(\textsc{RunEnc}^{\mathbb{A}}_{\mathcal{M}})$,
- $\textsc{RunEnc}_{\mathcal{M}} \supseteq \mathsf{proj}_{\mathbb{A}}(\textsc{RunEnc}^{\mathbb{A}}_{\mathcal{M}}) \supseteq \mathsf{proj}_{\mathbb{A}}(\textsc{RunEnc}^{\mathbb{A},<}_{\mathcal{M}})$.

which yields the corollaries:

Corollary 8.22. unordered data encodings $\equiv$ runs of $\mathcal{M}$

$\textsc{RunEnc}_{\mathcal{M}} = \mathsf{proj}_{\mathbb{A}}(\textsc{RunEnc}^{\mathbb{A}}_{\mathcal{M}})$.

Corollary 8.23. ordered data encodings $\equiv$ runs of $\mathcal{M}$

$\textsc{RunEnc}_{\mathcal{M}} = \mathsf{proj}_{\mathbb{A}}(\textsc{RunEnc}^{\mathbb{A},<}_{\mathcal{M}})$.

Proof of lemma 8.21.

As corollary 8.20 trivially implies that $\mathsf{proj}_{\mathbb{A}}(\textsc{RunEnc}^{\mathbb{A},<}_{\mathcal{M}}) \subseteq \mathsf{proj}_{\mathbb{A}}(\textsc{RunEnc}^{\mathbb{A}}_{\mathcal{M}})$, we are left with only two inclusion relations to prove.

$\textsc{RunEnc}_{\mathcal{M}} \supseteq \mathsf{proj}_{\mathbb{A}}(\textsc{RunEnc}^{\mathbb{A}}_{\mathcal{M}})$

Take any $w \in \textsc{RunEnc}^{\mathbb{A}}_{\mathcal{M}}$. As $w \in \textsc{Reg}^{\mathbb{A}}_{\mathcal{M}}$, we know that w corresponds to a sequence

$$[\![\mathsf{proj}_{\mathbb{A}}(w)]\!] = (p_1, \mu_1) \xleftarrow{\delta_1} (p_2, \mu_2) \xleftarrow{\delta_2} \dots \xleftarrow{\delta_{n-2}} (p_{n-1}, \mu_{n-1}) \xleftarrow{\delta_{n-1}} (p_n, \mu_n)$$

which is consistent with zero tests, the successive transition rules match, and it starts in a valid configuration. It remains to show that well-alignment implies that correct inequalities hold between every two consecutive valuations μ_1, μ_2. Fix arbitrary infix of w of the form

$$(\delta_0, \_)\; v_1\; ((p_2, \mathsf{op}, p_1), \_)\; v_2\; (\delta_2, \_).$$

For any counter $\mathtt{c} \in \mathtt{C}$, if it is not modified by op, condition 8.18.C implies that there are as many symbols $(\mathtt{c}, \_)$ in v_2 as there are distinct data values in v_1. In turn, condition 8.18.B implies that all data in v_1 are pairwise different, thus $\mu_1(\mathtt{c}) \leq \mu_2(\mathtt{c})$, as it was required. Other inequalities, for $\mathsf{op} \in \{\text{‘}\mathtt{c} \mathrel{+}= 1\text{’}, \text{‘}\mathtt{c} \mathrel{-}= 1\text{’}\}$ are handled similarly. Therefore, $[\![\mathsf{proj}_{\mathbb{A}}(w)]\!] \in \mathsf{Runs}(\mathcal{M})$.

$\textsc{RunEnc}_{\mathcal{M}} \subseteq \mathsf{proj}_{\mathbb{A}}(\textsc{RunEnc}^{\mathbb{A},<}_{\mathcal{M}})$

Let $w \in \textsc{RunEnc}_{\mathcal{M}}$. By definition, there exists $\pi \in \mathsf{Runs}(\mathcal{M})$ such that $w = \mathsf{enc}(\pi)$. We only need to show that each letter of $\mathsf{enc}(\pi)$ (as in eq. (8.2)), can be labeled with an atom, such that the resulting data word meets conditions 8.18.A, 8.18.B and 8.18.C$^<$.

Let us fix $\alpha \in \mathbb{A}$, which will be used as a label of all symbols from Δ'. Put $\mathbb{A}' = \mathbb{A} \setminus \{\alpha\}$. We construct the labeling of infixes $v_i = \mathsf{enc}(\mu_i)$ inductively. We start by pairing each letter of v_1 with pairwise-different atoms chosen arbitrarily from $\mathbb{A}'$.

Inductive step

Let us assume that letters of v_i have been already assigned atoms. We construct the labeling of v_{i+1} sequentially, in four steps, handling one individual counter of $\mathtt{c}_1, \mathtt{c}_2, \mathtt{c}_3, \mathtt{c}_4 \in \mathtt{C}$ at a time. Let $v_\bullet^j$ denote the (maximal) infixes of $v_\bullet$ corresponding to counter $\mathtt{c}_j$. In j-th step, depending on $\mathtt{op}_i$, we label v_i^j as follows:

a) If $\mathtt{op}_i$ does not modify $\mathtt{c}$, then $v_i^{\mathtt{c}} = \mu_i(\mathtt{c}) \leq \mu_{i+1}(\mathtt{c}) = v_{i+1}^{\mathtt{c}}$. We apply the labeling of $v_i^{\mathtt{c}}$ to the first $|\mu_i(\mathtt{c})|$ letters of $v_{i+1}^{\mathtt{c}}$. Remaining letters are labeled with pairwise-different and fresh[3] atoms from $\mathbb{A}'$.

b) If $\mathtt{op}_i =$ '$\mathtt{c} \mathrel{+}= 1$', when computing the labeling of $v_{i+1}^{\mathtt{c}}$, we disregard the first symbol of $v_i^{\mathtt{c}}$. The remaining atoms with which $v_i^{\mathtt{c}}$ was annotated are used to label $|\mu_i(\mathtt{c})| - 1$ letters of $v_{i+1}^{\mathtt{c}}$. As before, the remaining letters of $v_{i+1}^{\mathtt{c}}$ get fresh[3] atoms.

c) If $\mathtt{op}_i =$ '$\mathtt{c} \mathrel{-}= 1$', we label the first symbol of $v_{i+1}^{\mathtt{c}}$ with a fresh[3] atom, and handle the remaining part the same way as in case a).

Note that this construction guarantees that each v_i is labeled with pairwise-different atoms, thus satisfying condition 8.18.B. Condition 8.18.A is also trivially met. Finally, observe that the case enumeration above match the cases listed in condition 8.18.C$^<$. Therefore, one can easily show that condition 8.18.C$^<$ holds. □

8.7 Undecidability of DRA membership

The aim of this section is to prove

Theorem 8.3 DRA membership und.

The DRA membership problem is undecidable for NRA_1.

The reduction presented here has two main ingredients—the lemmas stated below.

Lemma 8.24. wrong data encodings are recognised by NRA_1

$(\Sigma_{\mathcal{M}} \times \mathbb{A})^* \setminus \textsc{RunEnc}_{\mathcal{M}}^{\mathbb{A}} \in \mathsf{NRA}_1$ and the automaton can be effectively constructed.

Lemma 8.25. $\mathcal{M}$'s finiteness equivalent to determinisability of $\textsc{RunEnc}_{\mathcal{M}}^{\mathbb{A}}$

The set of reachable configurations $\mathsf{ReachConf}(\mathcal{M})$ is finite if, and only if, $\textsc{RunEnc}_{\mathcal{M}}^{\mathbb{A}} \in$ DRA.

Proof of theorem 8.3.

The reduction from theorem 8.5 is a result of a trivial juxtaposition of lemmas 8.24 and 8.25: given LCM_4 $\mathcal{M}$, construct the automaton for $(\Sigma_{\mathcal{M}} \times \mathbb{A})^* \setminus \textsc{RunEnc}_{\mathcal{M}}^{\mathbb{A}}$ and ask about its determinisability. The answer will be positive if, and only if, the

[3] in particular, not used as labels for either v_i or any $v_{i_i}^k$, $k < j$

complement—$\textsf{RunEnc}^{\mathbb{A}}_{\mathcal{M}}$ —is a deterministic data language, which in turn occurs if, and only if, $\mathsf{ReachConf}(\mathcal{M})$ is finite. □

Proof of lemma 8.24.

Observe that

$$(\Sigma_{\mathcal{M}} \times \mathbb{A})^* \setminus \textsf{RunEnc}^{\mathbb{A}}_{\mathcal{M}} = \widehat{\textsf{Reg}}^{\mathbb{A}}_{\mathcal{M}} \cup \widehat{\mathsf{A}}^{\mathbb{A}}_{\mathcal{M}} \cup \widehat{\mathsf{B}}^{\mathbb{A}}_{\mathcal{M}} \cup \widehat{\mathsf{C}}^{\mathbb{A}}_{\mathcal{M}}.$$

Since the class of $\textsf{NRA}_1$ is effectively closed under unions, we only need to show that languages $\widehat{\mathsf{A}}^{\mathbb{A}}_{\mathcal{M}}$, $\widehat{\mathsf{B}}^{\mathbb{A}}_{\mathcal{M}}$, and $\widehat{\mathsf{C}}^{\mathbb{A}}_{\mathcal{M}}$ are recognised by $\textsf{NRA}_1$. Thus, the statement to be proven stems from the following lemmas 8.26 to 8.28. □

Lemma 8.26.

$\widehat{\mathsf{A}}^{\mathbb{A}}_{\mathcal{M}} \in \textsf{DRA}_1$.

Lemma 8.27.

$\widehat{\mathsf{B}}^{\mathbb{A}}_{\mathcal{M}} \in \textsf{NRA}_1$.

Lemma 8.28.

$\widehat{\mathsf{C}}^{\mathbb{A}}_{\mathcal{M}} \in \textsf{NRA}_1$.

Proof of lemma 8.26.

$\widehat{\mathsf{A}}^{\mathbb{A}}_{\mathcal{M}}$ is straightforwardly recognised by $\textsf{DRA}_1$ with one register $\mathtt{x}$ and just 3 states $L = \{\textsf{Ini}, \textsf{Ok}, \textsf{Err}\}$, first one being initial, $\textsf{Err}$ —accepting. For each $\sigma \in \Sigma_{\mathcal{M}}$ and $\delta \in \Delta'$, the automaton has transitions

$$(\textsf{Ini} \xRightarrow{\delta,\{\mathtt{x}\}} \textsf{Ok}) \qquad (\textsf{Ok} \xRightarrow{\delta,\mathtt{x}=\mathtt{y},\{\mathtt{x}\}} \textsf{Ok}) \qquad (\textsf{Ok} \xRightarrow{\sigma,\mathtt{x}\neq\mathtt{y}} \textsf{Ok})$$

The remaining transitions not listed above lead to the state $\textsf{Err}$. □

Proof of lemma 8.27.

Language $\widehat{\mathsf{B}}^{\mathbb{A}}_{\mathcal{M}}$ can be recognised by an $\textsf{NRA}$ with one register $\mathtt{x}$. The corresponding automaton simply guesses the position with a repeated atom $\alpha \in \mathbb{A}$ and goes to the state $\textsf{Err}$ upon finding superfluous copy of α within the same counter valuation encoding block. The simple construction is omitted. □

Proof of lemma 8.28.

As before, $\textsf{NRA}_1$ suffices to find a violation of condition 8.18.C. Here, the constructed automaton tries to locate within its input an infix w of the form

$$(\delta_1, \_)\ \underbrace{\ldots\ (\mathtt{c}, \alpha)\ \ldots}_{v_1}\ (\delta_2, \_)\ v_2\ (\delta_3, \_)$$

where $\delta_\bullet \in \Delta'$, $\delta_2 = (p, \mathsf{op}, q)$, $\mathtt{c} \in \mathtt{C}$, $\alpha \in \mathbb{A}$, which violates condition 8.18.C. One of the following cases

1. $\mathsf{op} \in \mathtt{Op} \setminus \{\text{`}\mathtt{c} \mathrel{+}= 1\text{'}, \text{`}\mathtt{c} \mathrel{-}= 1\text{'}\}$ and $(\mathtt{c}, \alpha)$ doest not appear in v_2,
2. $\mathsf{op} = \text{`}\mathtt{c} \mathrel{+}= 1\text{'}$ and $(\mathtt{c}, \alpha)$ does not appear in v_2, although it is not the first element in the sequence of symbols $\mathtt{c}$ in v_1,

3. $\mathtt{op} = $ '$\mathtt{c} \mathrel{-}= 1$' and $(\mathtt{c}, \alpha)$ does not appear as a non-first symbol of v_2.

Construction

We construct an automaton $\mathcal{E} \in \mathtt{NRA}_1$ with 23 states

$$S = \{\textsc{Ini}, \textsc{KeepPrev}, \textsc{Err}\} \cup (\{\textsc{KeepPrev}, \textsc{Sel}, \textsc{SelFst}, \textsc{SkipFst}, \textsc{Find}\} \times \mathtt{C}),$$

$\textsc{Ini}$ being initial, $\textsc{Err}$ —accepting. Elements $(\bullet, \mathtt{c})$ of the last component of the expression above are denoted as $\bullet(\mathtt{c})$ (e.g., $\textsc{KeepPrev}(\mathtt{c})$). The constructed device first skips an arbitrary prefix of the input word

$$(\textsc{Ini} \overset{\sigma}{\Rightarrow} \textsc{Ini}) \qquad \text{for } \sigma \in \Sigma_{\mathcal{M}}$$

and nondeterministically guesses the beginning of the infix w

$$(\textsc{Ini} \overset{\delta}{\Rightarrow} \textsc{KeepPrev}) \qquad \text{for } \sigma \in \Delta'.$$

Afterwards, it skips some prefix of v_1, keeping track of the most recently seen letter $\mathtt{b} \in \mathtt{C}$

$$(\textsc{KeepPrev} \overset{\mathtt{b}}{\Rightarrow} \textsc{KeepPrev}(\mathtt{b})) \qquad \text{for } \mathtt{b} \in \mathtt{C},$$

$$(\textsc{KeepPrev}(\mathtt{b}) \overset{\mathtt{c}}{\Rightarrow} \textsc{KeepPrev}(\mathtt{c})) \qquad \text{for } \mathtt{b}, \mathtt{c} \in \mathtt{C},$$

until it nondeterministically selects some symbol $(\mathtt{c}, \alpha)$ within v_1. When that occurs, atom α gets stored in the register $\mathtt{x}$, while the letter $\mathtt{c}$ is remembered in state $\textsc{SelFst}(\mathtt{c})$ or $\textsc{Sel}(\mathtt{c})$. The actual state depends on whether $\mathtt{c}$ was the first of a sequence of symbols $\mathtt{c}$ in v_1

$$(\textsc{KeepPrev} \xRightarrow{\mathtt{c},\mathtt{x}=\mathtt{y}} \textsc{SelFst}(\mathtt{c})) \qquad \text{for } \mathtt{c} \in \mathtt{C}$$

$$(\textsc{KeepPrev}(\mathtt{b}) \xRightarrow{\mathtt{c},\mathtt{x}=\mathtt{y}} \textsc{SelFst}(\mathtt{c})) \qquad \text{for } \mathtt{b}, \mathtt{c} \in \mathtt{C},\ \mathtt{b} \neq \mathtt{c}$$

or not (in which case $\textsc{Sel}$ is used in $\textsc{SelFst}$'s stead)

$$(\textsc{KeepPrev}(\mathtt{c}) \xRightarrow{\mathtt{c},\mathtt{x}=\mathtt{y}} \textsc{Sel}(\mathtt{c})) \qquad \text{for } \mathtt{c} \in \mathtt{C}.$$

State $\textsc{Sel}$ or $\textsc{SelFst}$ does not change while letters from $\mathtt{C}$ are being read

$$(\textsc{Sel}(\mathtt{c}) \overset{\mathtt{d}}{\Rightarrow} \textsc{Sel}(\mathtt{c})) \qquad \text{for } \mathtt{c}, \mathtt{d} \in \mathtt{C}$$

$$(\textsc{SelFst}(\mathtt{c}) \overset{\mathtt{d}}{\Rightarrow} \textsc{SelFst}(\mathtt{c})) \qquad \text{for } \mathtt{c}, \mathtt{d} \in \mathtt{C}.$$

Upon reading a symbol $\delta \in \Delta$, the automaton has collected enough information to decide whether, according to condition 8.18.C, the previously selected symbol needs to be repeated or not. In cases, where repetition of that symbol is required, $\mathcal{E}$ proceeds to the phase of searching for $(\mathtt{c}, \alpha)$ in appropriate part of v_2, possibly skipping its first symbol. Its transition relation, for every $\delta = (\_, \mathtt{op}, \_) \in \Delta$ and $\mathtt{c} \in \mathtt{C}$, features the following transitions

$$(\textsc{Sel}(\mathtt{c}) \overset{\delta}{\Rightarrow} \textsc{Find}(\mathtt{c})) \qquad \text{for } \mathtt{op} \in \mathtt{Op} \setminus \{\text{`}\mathtt{c} \mathrel{-}= 1\text{'}\},$$

$$(\textsc{SelFst}(\mathtt{c}) \overset{\delta}{\Rightarrow} \textsc{Find}(\mathtt{c})) \qquad \text{for } \mathtt{op} \in \mathtt{Op} \setminus \{\text{`}\mathtt{c} \mathrel{+}= 1\text{'}, \text{`}\mathtt{c} \mathrel{-}= 1\text{'}\},$$

$$(\textsc{Sel}(\mathtt{c}) \overset{\delta}{\Rightarrow} \textsc{SkipFst}(\mathtt{c})) \qquad \text{for } \mathtt{op} = \text{`}\mathtt{c} \mathrel{-}= 1\text{'},$$

$$(\textsc{SelFst}(\mathtt{c}) \overset{\delta}{\Rightarrow} \textsc{SkipFst}(\mathtt{c})) \qquad \text{for } \mathtt{op} = \text{`}\mathtt{c} \mathrel{-}= 1\text{'}.$$

In case when the first symbol of v_2 was to be skipped, the search begins right after reading it

$$(\textsf{SkipFst}(\mathtt{c}) \overset{\mathtt{d}}{\Rightarrow} \textsf{Find}(\mathtt{c})) \qquad \text{for } \mathtt{c}, \mathtt{d} \in \mathtt{C}.$$

The search continues while the symbols read in the process are either not equal to $(\mathtt{c}, \alpha)$

$$(\textsf{Find}(\mathtt{c}) \overset{\mathtt{d}}{\Rightarrow} \textsf{Find}(\mathtt{c})) \qquad \text{for } \mathtt{c}, \mathtt{d} \in \mathtt{C},\ \mathtt{c} \neq \mathtt{d},$$

$$(\textsf{Find}(\mathtt{c}) \xRightarrow{\mathtt{c}, \mathtt{x} \neq \mathtt{y}} \textsf{Find}(\mathtt{c})) \qquad \text{for } \mathtt{c} \in \mathtt{C},$$

or the next symbol from Δ' is encountered—witnessing an error

$$(\textsf{Find}(\mathtt{c}) \overset{\delta}{\Rightarrow} \textsf{Err}) \qquad \text{for } \mathtt{c} \in \mathtt{C}, \text{ and } \delta \in \Delta'$$

which is then remembered until the end of the input

$$(\textsf{Err} \overset{\sigma}{\Rightarrow} \textsf{Err}) \qquad \text{for } \sigma \in \Sigma_{\mathcal{M}}.$$

In case $(\mathtt{c}, \alpha)$ is found, the automaton does not have any available transition, consequently rejecting the input. By construction, $\mathcal{E}$ accepts if, and only if, the input word does not meet the condition 8.18.C. ☐

Proof of lemma 8.25.

Given $\mathcal{M} \in \textsf{LCM}_4$, we need to prove the equivalence

$$|\textsf{ReachConf}(\mathcal{M})| < \infty \iff \textsf{RunEnc}_{\mathcal{M}}^{\mathbb{A}} \in \textsf{DRA}.$$

The '⇒' implication

Assume that $\textsf{ReachConf}(\mathcal{M})$ is finite. Since the size of $\mathcal{M}$'s configuration space is limited, it becomes a finite state system and thus the language $\textsf{RunEnc}_{\mathcal{M}}$ is regular. However, to deterministically recognise $\textsf{RunEnc}_{\mathcal{M}}^{\mathbb{A}}$, we also need to verify conditions 8.18.A to 8.18.C. Let $k \in \mathbb{N}$ be the finite upper bound on the attainable sum of $\mathcal{M}$'s counter values. By corollary 8.22, in every $w \in \textsf{RunEnc}_{\mathcal{M}}^{\mathbb{A}}$ the length of infixes from $\textsf{ValEnc}_{\mathcal{M}}$is bounded by k, too. We will show that $\textsf{RunEnc}_{\mathcal{M}}^{\mathbb{A}}$ can be recognised by $\textsf{DRA}_{2k+2}$. Note that $\textsf{A}_{\mathcal{M}}^{\mathbb{A}} \in \textsf{DRA}_1$, since it is a complement of $\textsf{DRA}_1$ language (cf. lemma 8.26).

The '⇐' implication

Let us assume that $\textsf{ReachConf}(\mathcal{M})$ is infinite, and—towards contradiction—that $\textsf{RunEnc}_{\mathcal{M}}^{\mathbb{A}}$ is recognised by some $\mathcal{A} \in \textsf{DRA}$ with a finite set of registers $\mathtt{X}$, $k = |\mathtt{X}|$. Due to first assumption, there is no finite upper bound on the attainable counter values. In particular, there exists a run π ending with a counter valuation μ with $\mu(\mathtt{c}) = k + 2$. Due to corollary 8.22, π has some encoding $w \in \textsf{RunEnc}_{\mathcal{M}}^{\mathbb{A}}$, which needs to have the form

$$w = \underbrace{(\triangleright, \alpha)\ \overbrace{v'\ (\mathtt{c}, \beta_0) \ldots (\mathtt{c}, \beta_{k+1})\ v''}^{\textsf{enc}(\mu)}\ (\delta, \alpha)}_{w'}\ w'',$$

where v', v'' correspond to counters other than $\mathtt{c}$, and $\delta \in \Delta$. When $\mathcal{A}$ is reading w, after observing the $\mathsf{enc}(\mu)$ block, it must forget at least two atoms from $\{\beta_0, \beta_1, \ldots, \beta_{k+1}\}$. Consequently, at least one atom $\beta \in \{\beta_1, \ldots, \beta_{k+1}\}$ is not stored in registers. Recall condition 8.18.C. Observe that if the counter operation of δ is '$c \mathrel{+}= 1$', symbol $(\mathtt{c}, \beta_0)$ does not have to appear in w'' to meet the condition. However, the remaining k letters $(\mathtt{c}, \beta_1), \ldots, (\mathtt{c}, \beta_k)$ are required to reappear in the next valuation encoding.

Intuitively, we will arrive at a contradiction by showing that the letter $(\mathtt{c}, \beta)$ can be removed from ? without influencing the acceptance of $\mathcal{A}$.

The resulting encoding $w'\psi(w'')$ is no longer valid, since it violates condition 8.18.C, but $\mathcal{A}$ still accepts it. This contradicts that $\mathcal{A}$ recognises $\textsc{RunEnc}^{\mathbb{A}}_{\mathcal{M}}$, and thus $\textsc{RunEnc}^{\mathbb{A}}_{\mathcal{M}}$ is not a DRA language, as required. ☐

8.8 Undecidability of register synthesis

In this section, we continue to develop the tools that helped to show the undecidability of the DRA membership problem, aiming this time to prove the decision version of the register synthesis problem undecidable:

Theorem 8.4 synthesis of DRA und.

The register synthesis decision problem is undecidable, and this holds already when Alice's winning condition is an NRA_2 language.

For $\mathcal{M} \in \mathsf{LCM}_4$, we will define a register synthesis game $\mathsf{G}^{\mathbb{A}}_{\mathcal{M}}$ with condition $W_{\mathcal{M}}$ and show that:

Lemma 8.29. Alice's winning condition in $\mathsf{G}^{\mathbb{A}}_{\mathcal{M}}$ is NRA_2

$W_{\mathcal{M}} \in \mathsf{NRA}_2$.

Lemma 8.30. $\mathsf{G}^{\mathbb{A}}_{\mathcal{M}}$ characterises determinisability of $\textsc{RunEnc}^{\mathbb{A},<}_{\mathcal{M}}$

$\mathsf{ReachConf}(\mathcal{M})$ is finite if, and only if, there exists a winning strategy for Bob in $\mathsf{G}^{\mathbb{A}}_{\mathcal{M}}$.

Proof of theorem 8.4.

Undecidability is a trivial consequence of theorem 8.5 and lemmas 8.29 and 8.30. ☐

8.8.1 Local errors and local correctness

Below we define two NRA languages, $\widehat{\mathsf{B}}{\downarrow}^{\mathbb{A}}_{\mathcal{M}}$ and $\widehat{\mathsf{C}}{\downarrow}^{\mathbb{A},<}_{\mathcal{M}}$, which are 'localised' versions of $\widehat{\mathsf{B}}^{\mathbb{A}}_{\mathcal{M}}$ and $\widehat{\mathsf{C}}^{\mathbb{A},<}_{\mathcal{M}}$ defined in §8.6. More precisely, they have a property that for any $w \in (\Sigma_{\mathcal{M}} \times \mathbb{A})^*$

- $w \in \widehat{\mathsf{B}}^{\mathbb{A}}_{\mathcal{M}}$ if, and only if, there exists a prefix $v \sqsubseteq w$ such that $v \in \widehat{\mathsf{B}}{\downarrow}^{\mathbb{A}}_{\mathcal{M}}$,
- $w \in \widehat{\mathsf{C}}^{\mathbb{A},<}_{\mathcal{M}}$ if, and only if, there exists a prefix $v \sqsubseteq w$ such that $v \in \widehat{\mathsf{C}}{\downarrow}^{\mathbb{A},<}_{\mathcal{M}}$.

$\widehat{\mathsf{B}}{\downarrow}^{\mathbb{A}}_{\mathcal{M}}$ is simply the language of the automaton shown in the proof of lemma 8.27, with only the transition ($\textsc{Err} \Rightarrow \textsc{Err}$) removed from its transition relation.

The same would apply to $\widehat{\mathsf{C}}{\downarrow}^{\mathbb{A},<}_{\mathcal{M}}$ and $\widehat{\mathsf{C}}^{\mathbb{A},<}_{\mathcal{M}}$, but the latter language was not shown to be NRA. Therefore, we need to prove the following

8.8.2 Definition of $\mathsf{G}^{\mathbb{A}}_{\mathcal{M}}$

Players in game $\mathsf{G}^{\mathbb{A}}_{\mathcal{M}}$ have actions

$$A = \Sigma_{\mathcal{M}} \qquad \text{(Alice)}$$
$$B = \{\textsf{OK}, \rightsquigarrow, \textsf{ERR}\mathsf{R}, \textsf{ERR}\mathsf{A}, \textsf{ERR}\mathsf{B}, \textsf{ERR}\mathsf{C}\} \qquad \text{(Bob)}$$

Let us define $\mathsf{proj}_\mathsf{A} : A \cdot B \cdot \mathbb{A} \to A \cdot \mathbb{A}$ as the projection function which ignores Bob's letters: $\mathsf{proj}_\mathsf{A}(a, b, \alpha) = (a, \alpha)$. Complementarily, let $\mathsf{proj}_\mathsf{B} : A \cdot B \cdot \mathbb{A} \to B$ be a projection which ignores Alice's letters and data values: $\mathsf{proj}_\mathsf{B}(a, b, \alpha) = b$. We extend both functions homomorphically to finite and infinite words. The winning set of Alice is $W_{\mathcal{M}} = \textsc{Reach}(V^{\mathbb{A}}_{\mathcal{M}})$ for the following language $V^{\mathbb{A}}_{\mathcal{M}}$ of finite data words:

$$\begin{aligned}
V^{\mathbb{A}}_{\mathcal{M}} = {} & \mathsf{proj}_\mathsf{B}^{-1}(\textsf{OK}^* \rightsquigarrow^+ \textsf{OK}) && (8.14)\\
& \cup\, \mathsf{proj}_\mathsf{B}^{-1}(\textsf{OK}^* \rightsquigarrow^* \rightsquigarrow) \cap \mathsf{proj}_\mathsf{A}^{-1}(\textsc{Reg}^{\mathbb{A}}_{\mathcal{M}}) && (8.15)\\
& \cup\, \mathsf{proj}_\mathsf{B}^{-1}(\textsf{OK}^* \rightsquigarrow^* \textsf{ERR}\mathsf{R}) \cap \mathsf{proj}_\mathsf{A}^{-1}(\mathsf{R}^{\mathbb{A}}_{\mathcal{M}}) && (8.16)\\
& \cup\, \mathsf{proj}_\mathsf{B}^{-1}(\textsf{OK}^* \rightsquigarrow^* \textsf{ERR}\mathsf{A}) \cap \mathsf{proj}_\mathsf{A}^{-1}(\mathsf{A}^{\mathbb{A}}_{\mathcal{M}} \cup \widehat{\mathsf{R}}^{\mathbb{A}}_{\mathcal{M}}) && (8.17)\\
& \cup\, \mathsf{proj}_\mathsf{B}^{-1}(\textsf{OK}^* \rightsquigarrow^* \textsf{ERR}\mathsf{B}) \cap \mathsf{proj}_\mathsf{A}^{-1}(\mathsf{B}{\downarrow}^{\mathbb{A}}_{\mathcal{M}} \cup \widehat{\mathsf{R}}^{\mathbb{A}}_{\mathcal{M}} \cup \widehat{\mathsf{A}}^{\mathbb{A}}_{\mathcal{M}}) && (8.18)\\
& \cup\, \mathsf{proj}_\mathsf{B}^{-1}(\textsf{OK}^* \rightsquigarrow^* \textsf{ERR}\mathsf{C}) \cap \mathsf{proj}_\mathsf{A}^{-1}(\mathsf{C}{\downarrow}^{\mathbb{A},<}_{\mathcal{M}} \cup \widehat{\mathsf{R}}^{\mathbb{A}}_{\mathcal{M}} \cup \widehat{\mathsf{A}}^{\mathbb{A}}_{\mathcal{M}} \cup \widehat{\mathsf{B}}^{\mathbb{A}}_{\mathcal{M}}) && (8.19)\\
& \cup\, \mathsf{proj}_\mathsf{B}^{-1}(\textsf{OK}^* \textsf{OK}) \cap \mathsf{proj}_\mathsf{A}^{-1}(\widehat{\mathsf{R}}^{\mathbb{A}}_{\mathcal{M}} \cup \widehat{\mathsf{A}}^{\mathbb{A}}_{\mathcal{M}} \cup \widehat{\mathsf{B}}^{\mathbb{A}}_{\mathcal{M}} \cup \widehat{\mathsf{C}}^{\mathbb{A},<}_{\mathcal{M}}) && (8.20)
\end{aligned}$$

Intuition

It cannot go unnoticed that the above definition bears a striking resemblance to one of $V^{\mathbb{T}}_{\mathcal{M}}$, which appears in §8.5. It is important to point out that though the semantics of the languages appearing in both definitions differs, the high level mechanics of the game remains the same. More precisely, the entire content of the §8.5.2 applies to $\mathsf{G}^{\mathbb{A}}_{\mathcal{M}}$, provided that the following substitutions are made in its text

$$\begin{array}{llll}
\mathsf{G}^{\mathbb{T}}_{\mathcal{M}} \mapsto \mathsf{G}^{\mathbb{A}}_{\mathcal{M}} & V^{\mathbb{T}}_{\mathcal{M}} \mapsto V^{\mathbb{A}}_{\mathcal{M}} & \textsc{RunEnc}^{\mathbb{T}}_{\mathcal{M}} \mapsto \textsc{RunEnc}^{\mathbb{A}}_{\mathcal{M}} & \text{timed word} \mapsto \text{data word} \\
\mathsf{R}^{\mathbb{T}}_{\mathcal{M}} \mapsto \mathsf{R}^{\mathbb{A}}_{\mathcal{M}} & \mathsf{A}^{\mathbb{T}}_{\mathcal{M}} \mapsto \mathsf{A}^{\mathbb{A}}_{\mathcal{M}} & \mathsf{B}^{\mathbb{T}}_{\mathcal{M}} \mapsto \mathsf{B}{\downarrow}^{\mathbb{A}}_{\mathcal{M}} & \mathsf{C}{\downarrow}^{\mathbb{T}}_{\mathcal{M}} \mapsto \mathsf{C}{\downarrow}^{\mathbb{A},<}_{\mathcal{M}} \\
\widehat{\mathsf{R}}^{\mathbb{T}}_{\mathcal{M}} \mapsto \widehat{\mathsf{R}}^{\mathbb{A}}_{\mathcal{M}} & \widehat{\mathsf{A}}^{\mathbb{T}}_{\mathcal{M}} \mapsto \widehat{\mathsf{A}}^{\mathbb{A}}_{\mathcal{M}} & \widehat{\mathsf{B}}^{\mathbb{T}}_{\mathcal{M}} \mapsto \widehat{\mathsf{B}}^{\mathbb{A}}_{\mathcal{M}} & \widehat{\mathsf{C}}^{\mathbb{T}}_{\mathcal{M}} \mapsto \widehat{\mathsf{C}}^{\mathbb{A},<}_{\mathcal{M}}.
\end{array}$$

For this reason, we refrain from reproducing its contents here, instead restating only the 'register' version of lemma 8.17 below.

For a play $w \in (A \cdot B \times \mathbb{A})^{\omega}$, let $\mathsf{cut}(w) \in (A \cdot B \times \mathbb{A})^{*}$ denote its prefix up to and including the first symbol of the form $(\triangleleft, \_)$ if it exists, and an empty prefix ε otherwise. The winning condition is designed to ensure the following property:

Lemma 8.31. correspondence between $\textsc{RunEnc}^{\mathbb{A},<}_{\mathcal{M}}$ and plays of $\mathsf{G}^{\mathbb{A}}_{\mathcal{M}}$

Consider a play $w \in \mathsf{Plays}(\mathsf{G}^{\mathbb{A}}_{\mathcal{M}}) \setminus W_{\mathcal{M}}$ with nonempty prefix $v = \mathsf{cut}(w)$. Then

$$\mathsf{proj}_{\mathsf{A}}(v) \in \textsc{RunEnc}^{\mathbb{A},<}_{\mathcal{M}} \iff \mathsf{proj}_{\mathsf{B}}(v) \in \textsc{ok}^{*}.$$

In words, if w is winning for Bob and $\mathsf{cut}(w)$ is not empty, then $\mathsf{proj}_{\mathsf{A}}(v)$ is a valid encoding of $\mathcal{M}$'s run if, and only if, Bob played only $\textsc{ok}$ in v. This characterisation will play a crucial role in proving lemma 8.30.

8.8.3 Proofs

The remainder of this section contains proofs of lemmas 8.29 to 8.31.

Proof of lemma 8.29.

Observe that due to it suffices to prove that $V^{\mathbb{A}}_{\mathcal{M}} \in \mathsf{NRA}_2$. Since the class of NRA_2 is closed with respect to unions, we only need to show that each of the languages stated in lines (8.5) – (8.11) is NRA_2.

This is evident for all lines but X, Y and Z, because of the languages Q, R and S. We show that in subsequent . ☐

Lemma 8.32.

$\widehat{\mathsf{C}}^{\mathbb{A},<}_{\mathcal{M}} \in \mathsf{NRA}_2$

Lemma 8.33. local validity of condition 8.18.B is in NRA^{g}_{1}

$\mathsf{B}{\downarrow}^{\mathbb{A}}_{\mathcal{M}} \in \mathsf{NRA}^{g}_{1}$

Note that this is the only component of the definition of $V^{\mathbb{A}}_{\mathcal{M}}$ which requires guessing to be recognised. A way of eliminating guessing is discussed in §8.8.4.

Lemma 8.34. local validity of condition 8.18.$\mathsf{C}^{<}$ is in $\textsf{NRA}_2$

$\mathsf{C}{\downarrow}^{\mathbb{A},<}_{\mathcal{M}} \in \textsf{NRA}$

Proof of lemma 8.30.

The '$\Rightarrow$' implication

Assume that $\mathsf{ReachConf}(M)$ is finite. There is some k such that every reachable configuration (p, μ) has size $\sum_{\mathsf{c}\in\mathsf{C}} \mu(\mathsf{c}) \leq k$. In this case, the set of correct data encodings of runs of M can be recognised by a $\textsf{DRA}_{(k+2)}$ A which resets clock $\mathtt{x}_j$ when reading the j-th position of block $p_i u_i \delta_i$ (which is of length $\leq k+2$). From A we can produce a winning controller for Bob with k clocks:

The '$\Leftarrow$' implication

Assume that $\mathsf{ReachConf}(\mathcal{M})$ is infinite and, towards reaching a contradiction, that Bob has a winning controller $\mathcal{B} = (\mathtt{X}, A, B, L, l_{\mathsf{ini}}, \Delta)$ with $k \in \mathbb{N}$ clocks. We will focus on constructing $\mathcal{B}' \in \textsf{DRA}_k$ recognising $\textsc{RunEnc}^{\mathbb{A},<}_{\mathcal{M}}$, which will imply by lemma 8.25 that $\mathsf{ReachConf}(\mathcal{M})$ is finite.

Construction of $\textsf{DRA}_k$

Let $\mathcal{B}' = (\mathtt{X}, A, L \cup \{\textsc{Ok}, \textsc{Err}\}, l_{\mathsf{ini}}, \{\textsc{Ok}\}, \Delta')$, where Δ' contains transitions

- $(p \xRightarrow{a,\varphi,\mathtt{Y}} q)$ for every $(p \xRightarrow{a/\textsf{OK},\varphi,\mathtt{Y}} q) \in \Delta$ where $a \in A \setminus \{\triangleleft\}$,
- $(p \xRightarrow{a,\varphi,\mathtt{Y}} \textsc{Ok})$ for every $(p \xRightarrow{\triangleleft/\textsf{OK},\varphi,\mathtt{Y}} q) \in \Delta$,
- $(p \xRightarrow{a,\varphi,\mathtt{Y}} \textsc{Err})$ for every $(p \xRightarrow{a/e,\varphi,\mathtt{Y}} q) \in \Delta$, where $e \in B \setminus \{\textsf{OK}\}$,
- $(\textsc{Err} \xRightarrow{a} \textsc{Err})$ for every $a \in A$.

Correctness

We need to show that $\mathcal{L}(\mathcal{B}') = \textsc{RunEnc}^{\mathbb{A},<}_{\mathcal{M}}$. Take any $u \in (A \times \mathbb{A})^*$. Let w be any $(u, \mathcal{B})$-conformant play of $\mathsf{G}^{\mathbb{A}}_{\mathcal{M}}$. As $\mathcal{B}$ is a winning controller, w is winning for Bob. Let $v = \mathsf{cut}(w)$. Note that by construction $\mathsf{proj}_{\mathsf{A}}(v) = u$, and v corresponds to an accepting run of $\mathcal{B}'$ if, and only if, $\mathsf{proj}_{\mathsf{B}}(v) \in \textsf{OK}^*$. Therefore we have

$$u \in \mathcal{L}(\mathcal{B}') \iff \mathsf{proj}_{\mathsf{B}}(v) \in \textsf{OK}^* \iff \mathsf{proj}_{\mathsf{A}}(v) = u \in \textsc{RunEnc}^{\mathbb{A},<}_{\mathcal{M}}$$

where the last equivalence follows from lemma 8.31. ☐

Proof of lemma 8.31.

Let us fix a play $w \in \mathsf{Plays}(\mathsf{G}^{\mathbb{A}}_{\mathcal{M}}) \setminus W_{\mathcal{M}}$ winning for Bob, such that $v = \mathsf{cut}(w)$ is not empty. We need to show that

$$\mathsf{proj}_{\mathsf{A}}(v) \in \textsc{RunEnc}^{\mathbb{A},<}_{\mathcal{M}} \iff \mathsf{proj}_{\mathsf{B}}(v) \in \textsf{OK}^*$$

The '$\Rightarrow$' implication

Assume $\mathsf{proj}_{\mathsf{A}}(v) \in \textsf{RunEnc}^{\mathbb{A},<}_{\mathcal{M}} = \textsf{Reg}^{\mathbb{A}}_{\mathcal{M}} \cap \mathsf{A}^{\mathbb{A}}_{\mathcal{M}} \cap \mathsf{B}^{\mathbb{A}}_{\mathcal{M}} \cap \mathsf{C}^{\mathbb{A},<}_{\mathcal{M}}$. Towards contradiction, assume that $\mathsf{proj}_{\mathsf{B}}(v) \notin \textsf{ok}^*$. We aim at showing that w is not winning for Bob. Note that it suffices to exhibit a prefix $v' \sqsubseteq v$ belonging to $V^{\mathbb{A}}_{\mathcal{M}}$. Let $E = \{\textsf{err}\mathsf{R}, \textsf{err}\mathsf{A}, \textsf{err}\mathsf{B}, \textsf{err}\mathsf{C}\}$.

Case 1: $\mathsf{proj}_{\mathsf{B}}(v) \in B^*EB^*$

Let $v' \sqsubseteq v$ be the shortest prefix of v such that the last letter b of $\mathsf{proj}_{\mathsf{B}}(v')$ belongs to E. Since w is winning, $\mathsf{proj}_{\mathsf{B}}(v') \in \textsf{ok}^*{\rightsquigarrow}^*E$ (as $\textsf{ok}$ cannot follow $\rightsquigarrow$). Note that $\mathsf{proj}_{\mathsf{A}}(v') \in \mathsf{R}^{\mathbb{A}}_{\mathcal{M}} \cap \mathsf{A}^{\mathbb{A}}_{\mathcal{M}} \cap \mathsf{B}^{\mathbb{A}}_{\mathcal{M}} \cap \mathsf{C}{\downarrow}^{\mathbb{T}}_{\mathcal{M}}$, because

- $\mathsf{R}^{\mathbb{A}}_{\mathcal{M}}$ is the prefix closure of $\textsf{Reg}^{\mathbb{A}}_{\mathcal{M}}$,
- languages $\mathsf{A}^{\mathbb{A}}_{\mathcal{M}}$ and $\mathsf{B}^{\mathbb{A}}_{\mathcal{M}}$ are both prefix-closed, and
- $\mathsf{proj}_{\mathsf{A}}(v') \in \mathsf{C}{\downarrow}^{\mathbb{T}}_{\mathcal{M}}$ due to fact 8.14.

therefore, depending on the letter $b \in E$, one of the components (8.7) – (8.10) matches v', implying $v' \in V^{\mathbb{A}}_{\mathcal{M}}$.

Case 2: $\mathsf{proj}_{\mathsf{B}}(v) \in \{\textsf{ok}, \rightsquigarrow\}^*$

In this case $\mathsf{proj}_{\mathsf{B}}(v)$ needs to contain symbol $\rightsquigarrow$. If it is followed by $\textsf{ok}$ symbol, some prefix of v trivially matches the component (8.5) of $v_{\mathcal{M}}$'s definition. Otherwise, $\mathsf{proj}_{\mathsf{B}}(v) \in \textsf{ok}^*{\rightsquigarrow}^+$. But $\mathsf{proj}_{\mathsf{A}}(v) \in \textsf{Reg}^{\mathbb{A}}_{\mathcal{M}}$ and therefore $v \in V^{\mathbb{A}}_{\mathcal{M}}$ (cf. (8.6)).

The '$\Leftarrow$' implication

Assume that $\mathsf{proj}_{\mathsf{B}}(v) \in \textsf{ok}^*$. Since w is winning, no prefix of v belongs to $V^{\mathbb{A}}_{\mathcal{M}}$. Therefore, for every $v' \sqsubseteq v$ we have $\mathsf{proj}_{\mathsf{A}}(v') \in \mathsf{R}^{\mathbb{A}}_{\mathcal{M}} \cap \mathsf{A}^{\mathbb{A}}_{\mathcal{M}} \cap \mathsf{B}^{\mathbb{A}}_{\mathcal{M}} \cap \mathsf{C}{\downarrow}^{\mathbb{A},<}_{\mathcal{M}}$ (cf. (8.11)). Using fact 8.14 we conclude that $\mathsf{proj}_{\mathsf{A}}(v) \in \mathsf{C}^{\mathbb{A},<}_{\mathcal{M}}$, and—consequently—$\mathsf{proj}_{\mathsf{A}}(v) \in \textsf{RunEnc}^{\mathbb{A},<}_{\mathcal{M}}$. $\square$

8.8.4 Eliminating guessing

The game we presented can be transformed to one without *guessing*. The construction, though conceptually easy, would add another layer of difficulty to the construction, therefore it was factored out. Here, we present the set of modifications needed to eliminate guessing.

Observe that in the definition of language V_{cM}, the only component which requires guessing is the language $\mathsf{B}^{\mathbb{A}}_{\mathcal{M}}$. The modified game will have a new auxiliary initial phase, with # marking the transition between phases.

Define $V'_{\mathcal{M}}$ analogously to $V_{\mathcal{M}}$, but with two differences

1. with $\mathsf{B}^{\mathbb{A}}_{\mathcal{M}}$ replaced by
2. with flc skcjrnpwesrcnl.

Chapter 9

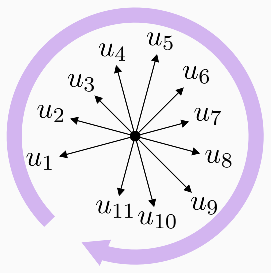
Utility results for VASS$_2$

The area of interest of the last chapter diverges slightly from the preceding parts, which are essentially devoted to register and timed automata.

9.1 Supporting definition – sequential cones

For a vector $v \in \mathbb{Z}^2$, define the half-line induced by v as $\ell_v := \mathbb{R}_{\geq 0} \cdot v = \{\alpha v \mid \alpha \in \mathbb{R}_{\geq 0}\}$. We call two vectors v, w *collinear* if $\ell_v = \ell_w$, and *contralinear* if $\ell_v = \ell_{-w}$. For two vectors $u, v \in \mathbb{Z}^2 \setminus \{(0,0)\}$, define the *angle* $\measuredangle[u,v] \subseteq \mathbb{R}^2$ as the union of all half-lines which lie clock-wise between ℓ_u and ℓ_v, including the two half-lines themselves. In particular, $\measuredangle[v,v] = \ell_v$. Analogously we define the sets $\measuredangle[u,v)$, $\measuredangle(u,v]$ and $\measuredangle(u,v)$ which exclude one or both of the half-lines. We refer to an angle of the form $\measuredangle[v,-v]$ as *half-plane*. We write $v \circlearrowright u$ when $u \in \measuredangle(v,-v)$, i.e., u is oriented clock-wise with respect to v (see figure 9.1 for an illustration). Note that $\circlearrowright$ defines a total order on pairwise non-collinear non-negative vectors.

Figure 9.1. Quasi-order $\circlearrowright$



Above $u_1 \circlearrowright u_2 \circlearrowright \ldots \circlearrowright u_{11} \circlearrowright u_1$. Also, $u_4 \circlearrowright u_9$, but $u_4 \not\circlearrowright u_{11}$ and $u_{11} \circlearrowright u_4$. Pairs of vectors u_i, u_{i+6} are contralinear, for $i = 1, \ldots, 5$.

By the *cone* of a finite set of vectors $\{v_1, \ldots, v_k\} \subseteq \mathbb{Z}^2$ we mean the set of all non-negative rational linear combinations of these vectors:

$$\textsc{Cone}(v_1, \ldots, v_k) := \{\Sigma_{j=1}^{k} a_j v_j \in \mathbb{R}^2 \mid a_1, \ldots, a_k \in \mathbb{R}_{\geq 0}\}.$$

We call the cone of a single vector $\textsc{Cone}(v) = \ell_v$ *trivial*, and the cone of zero vectors $\textsc{Cone}(\varnothing) = \{(0,0)\}$ *degenerate*. Two non-zero vectors v_1 and v_2 can be in four distinct relations: (I) they are collinear, (II) they are contralinear, (III) $v_1 \circlearrowright v_2$ and hence $\textsc{Cone}(v_1, v_2) = \measuredangle[v_1, v_2]$, (IV) $v_2 \circlearrowright v_1$ and hence $\textsc{Cone}(v_1, v_2) = \measuredangle[v_2, v_1]$.

Lemma 9.2.

Every cone either equals the whole plane $\mathbb{R}^2$, or is included in some half-plane.

Proof of lemma 9.2.

Assume, w.l.o.g. that the vectors $v_1, \ldots, v_k$ are non-zero and include no collinear pair. Suppose there is a contralinear pair v_i, v_j among $v_1, \ldots, v_k$. If all other vectors v_h satisfy $v_i \circlearrowleft v_h \circlearrowleft v_j$ then $\textsc{Cone}(v_1, \ldots, v_k)$ is included in the half-plane $\measuredangle[v_i, v_j]$. Otherwise $\textsc{Cone}(v_1, \ldots, v_k)$ is the whole plane.

Now suppose there is no contralinear pair among $v_1, \ldots, v_k$. If some three v_i, v_j, v_h of them satisfy $v_i \circlearrowleft v_j \circlearrowleft v_h \circlearrowleft v_i$ then $\textsc{Cone}(v_1, \ldots, v_k)$ includes the three angles $\measuredangle[v_i, v_j]$, $\measuredangle[v_j, v_h]$ and $\measuredangle[v_h, v_i]$, the union of which is the whole plane. Otherwise, the relation $\circlearrowleft$ is transitive and hence defines a (strict) total order on $\{v_1, \ldots, v_k\}$. The minimal and maximal element v_i and v_j w.r.t. the order satisfy $v_i \circlearrowleft v_j$, and hence $\textsc{Cone}(v_1, \ldots, v_k) = \measuredangle[v_i, v_j]$ is included in the half-plane $\measuredangle[v_i, -v_i]$. □

The *sequential cone* of vectors $v_1, \ldots, v_k \in \mathbb{Z}^2$ imposes additional non-negativeness conditions, namely for every i, the partial sum $a_1 v_1 + \ldots + a_i v_i$ must be non-negative (this is required later, when pumping cycles in a run whose effects are $v_1, \ldots, v_k$ in that order):

$$\textsc{SeqCone}(v_1, \ldots, v_k) := \{\Sigma_{j=1}^{k} a_j v_j \in \mathbb{R}^2_{\geq 0} \mid a_1, \ldots, a_k \in \mathbb{R}_{\geq 0}, \forall i \,.\, \Sigma_{j=1}^{i} a_j v_j \in \mathbb{R}^2_{\geq 0}\}.$$

Note that v_1 may be assumed w.l.o.g. to be semi-positive, but other vectors v_i are not necessarily non-negative; and that every sequential cone is a subset of the non-negative orthant $\mathbb{R}^2_{\geq 0}$. Importantly, contrarily to cones, the order of vectors $v_1, \ldots, v_k$ matters for sequential cones. In fact, sequential cones are just convenient syntactic sugar for cones of pairs of non-negative vectors:

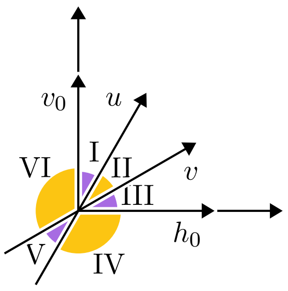
Lemma 9.3.

For all vectors $v_1, \ldots, v_k$, the sequential cone $\textsc{SeqCone}(v_1, \ldots, v_k)$, if not degenerate, equals $\textsc{Cone}(u, v)$, for two non-negative vectors u, v, and each of them either belongs to $\{v_1, \ldots, v_k\}$, or is horizontal, or vertical.

Proof of lemma 9.3.

We proceed by induction on k. For $k = 1$ we have $\textsc{SeqCone}(v_1) = \ell_{v_1} = \textsc{Cone}(v_1, v_1)$. Let v_0 and h_0 denote some fixed vertical and horizontal vector, respectively. For the induction step we assume $\textsc{SeqCone}(v_1, \ldots, v_{k-1}) = \textsc{Cone}(u, v)$ for non-negative vectors u, v; and compute the value of $\textsc{SeqCone}(v_1, \ldots, v_k)$, separately in each of the following distinct cases (assume w.l.o.g. $u \circlearrowleft v$):

$$\textsf{SeqCone}(v_1,\dots,v_k) = \begin{cases} \textsf{Cone}(v_k, v) & \text{if } v_k \in \measuredangle[v_0, u) \\ \textsf{Cone}(u, v) & \text{if } v_k \in \measuredangle[u, v] \\ \textsf{Cone}(u, v_k) & \text{if } v_k \in \measuredangle(v, h_0] \\ \textsf{Cone}(u, h_0) & \text{if } v_k \in \measuredangle(h_0, -u] \\ \textsf{Cone}(v_0, h_0) & \text{if } v_k \in \measuredangle(-u, -v) \\ \textsf{Cone}(v_0, v) & \text{if } v_k \in \measuredangle[-v, v_0). \end{cases}$$



□

9.2 Thin/thick dichotomy of runs

The main result of this section (cf. theorem 9.7 below) classifies $(0,0)$-runs in a VASS$_2$ into *thin* and *thick* ones. Throughout this section we consider an arbitrary fixed VASS$_2$ $V = (Q, T)$. Let $n = |Q|$ and $M = \|V\|$.

Thin runs

The *belt* of *direction* $v \in \mathbb{N}^2$ and *width* W is the set

$$\mathcal{B}_{v,W} = \{u \in \mathbb{N}^2 \mid \mathsf{dist}(u, \ell_v) \le W\},$$

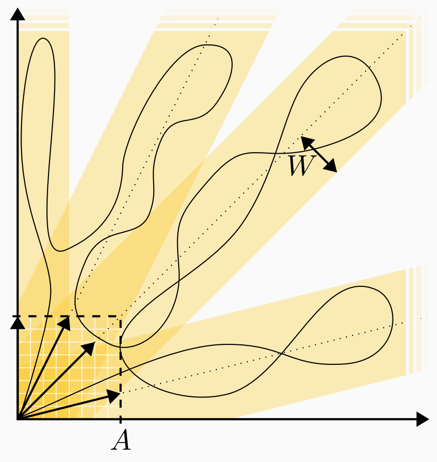
where $\mathsf{dist}(u, \ell_v)$ denotes the Euclidean distance between the point u and the half-line ℓ_v.

For $A \in \mathbb{N}$, we call $\mathcal{B}_{v,W}$ an *A-belt* if $\|v\| \le A$ and $W \le A$.

We say that a run ρ of V is *A-thin* if for every configuration c in ρ there exists an A-belt B such that $c \in Q \times B$.

Figure 9.4 shows an example of an A-thin run contained within four belts $\mathcal{B}_{v_i,W}$.

Figure 9.4. Thin run

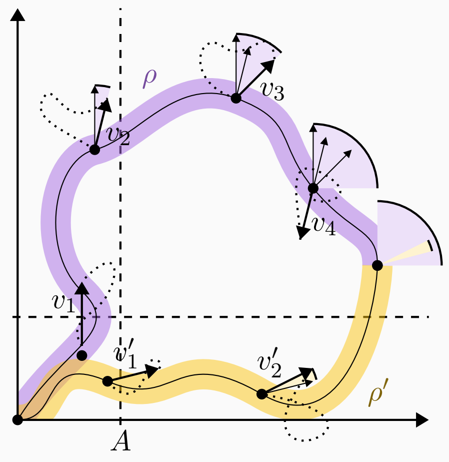


Thick runs

Let $A \in \mathbb{N}$. Four cycles $\pi_1, \pi_2, \pi_3, \pi_4 \in T^*$ are *A-sequentially enabled* in a run ρ if their lengths are at most A, and the run ρ factors into $\rho = \rho_1\,\rho_2\,\rho_3\,\rho_4\,\rho_5$ so that (denote by v_1, v_2, v_3, v_4 the effects of $\pi_1, \pi_2, \pi_3, \pi_4$, respectively):

- ▶ The effect v_1 is semi-positive, the cycle π_1 is enabled in $c_1 := \mathsf{trg}(\rho_1)$, and both coordinates are bounded by A along ρ_1.
- ▶ If v_1 is positive then π_2 is $\varnothing$-enabled in $c_2 := \mathsf{trg}(\rho_2)$. Otherwise (let j be the coordinate s.t. $v_1[j] = 0$) π_2 is $\{j\}$-enabled in $c_2 := \mathsf{trg}(\rho_2)$, and jth coordinate is bounded by A along ρ_2.
- ▶ The cycle π_i is $\varnothing$-enabled in $c_i := \mathsf{trg}(\rho_i)$, for $i = 3, 4$.

Figure 9.5. Thick run



We also say that the four vectors v_1, v_2, v_3, v_4 are A-sequentially enabled in ρ, quantifying the cycles existentially. A $(0,0)$-run τ is called *A-thick* if it partitions into $\tau = \rho\,\rho'$ so that

1. some vectors v_1, v_2, v_3, v_4 are A-sequentially enabled in ρ,

2. some vectors v'_1, v'_2, v'_3, v'_4 are A-sequentially enabled in $\mathsf{rev}(\rho')$,

3. $\textsc{SeqCone}(v_1, v_2, v_3, v_4) \cap \textsc{SeqCone}(v'_1, v'_2, v'_3, v'_4)$ is non-trivial.

Example 9.6. geometry of a thick cone

Figure 9.5 illustrates the geometric ideas underlying these three conditions for A-thick runs. Purple angles denote sequential cones $\textsc{SeqCone}(v_1, v_2)$, $\textsc{SeqCone}(v_1, v_2, v_3)$ and $\textsc{SeqCone}(v_1, v_2, v_3, v_4)$, respectively, and yellow angle denotes $\textsc{SeqCone}(v'_1, v'_2)$.

Concerning condition 1, a cycle π_1 depicted by a dotted line, with vertical effect v_1, can be used to increase the second (vertical) coordinate arbitrarily, which justifies the relaxed requirement that a cycle π_2 with effect v_2 is only $\{1\}$-enabled. Note that the norm of the configuration enabling π_1, as well as the first coordinate of the configuration enabling π_2, are bounded by A.

Concerning condition 2, a cycle π'_1 with positive effect v'_1 can be used to increase both coordinates arbitrarily; therefore a cycle π'_2 with effect v'_2 is only required to be $\varnothing$-enabled, and no coordinate of the configuration enabling π'_2 is required to be bounded by A. In the illustrated example, vectors v'_3 and v'_4 are not needed; formally, one can assume $v'_2 = v'_3 = v'_4$ and $\rho'_3 = \rho'_4 = \varepsilon$.

Condition 3 ensures that the cycles $\pi_1, \ldots, \pi_4$ and $\pi'_1, \ldots, \pi'_4$ can be pumped such that the pumped versions of ρ and ρ' are still connected.

In the illustrated example, observe that $\textsc{SeqCone}(v_1, v_2) \cap \textsc{SeqCone}(v'_1) = \varnothing$. Intuitively, both coordinates in the target of ρ can be increased arbitrarily using v_1 and v_2, and similarly both coordinates of the target of $\mathsf{rev}(\rho')$ can be increased arbitrarily using v'_1, but 'directions of increase' are non-crossing. Adding v_3 and v'_2 is not sufficient, as still $\textsc{SeqCone}(v_1, v_2, v_3) \cap \textsc{SeqCone}(v'_1, v'_2) = \varnothing$. When vector v_4 is adjoined, condition 3 holds as $\textsc{SeqCone}(v_1, v_2, v_3, v_4) = \mathbb{R}^2_{\geq 0}$. Finally, the four vectors are really needed here, e.g., vector v_3 can not be omitted as $\textsc{SeqCone}(v_1, v_2, v_4) = \textsc{SeqCone}(v_1, v_2)$.

The first of two main results in this chapter is:

Theorem 9.7. thin/thick dichotomy

There is a polynomial p such that every $(0, 0)$-run in a VASS_2 V is either $p(nM)^n$-thin or $p(nM)^n$-thick.

In its proof we use the following core fact:

Lemma 9.8. non-negative Cycle Lemma

There is a polynomial P such that every run ρ in V from a $(0, 0)$-configuration to a target configuration of norm larger than $P(nM)^n$, contains a configuration enabling a semi-positive cycle of length at most $P(nM)$.

Both theorem 9.7 and lemma 9.8 have relatively complex proofs; for this reason we place each proof in a separate subsection.

9.2.1 Proof of theorem 9.7

Proof of theorem 9.7.

Let P be the polynomial from lemma 9.8. The polynomial p required in theorem 9.7 can be chosen arbitrarily as long as $p(x) \geq \sqrt{2} \cdot \big(P(x) + (x+1)^3\big) \cdot x$. for all x; note that the following inequality follows:

$$p(nM)^n \geq \sqrt{2} \cdot \big((P(nM))^n + (nM+1)^3\big) \cdot nM. \tag{9.1}$$

In the sequel we deliberately confuse configurations $c = (q, v)$ with *their vectors v*: whenever convenient, we use c to denote the vector v, hoping that this does not lead to any confusion.

Let τ be a $(0,0)$-run of V which is not $p(nM)^n$-thin, i.e., τ contains therefore a configuration t which lies outside all the $p(nM)^n$-belts. We need to demonstrate points 1–3 in the definition of thick run. To this aim we split τ into $\tau = \rho\,\rho'$ where $\mathsf{trg}(\rho) = t = \mathsf{src}(\rho')$, and are going to prove the following two claims (a) and (a'). Let $D := P(nM)^n + (nM+1)^3$. For $x, y \in \mathbb{R}^2$, let $\mathsf{dist}(x,y)$ denote their Euclidean distance.

(a) Some vectors v_1, v_2, v_3, v_4 are $P(nM)^n$-sequentially enabled in ρ, and the sequential cone $\textsc{SeqCone}(v_1, v_2, v_3, v_4)$ contains a point $u \in \mathbb{R}^2_{\geq 0}$ with $\|u - t\| \leq D$.

(a') Some vectors v'_1, v'_2, v'_3, v'_4 are $P(nM)^n$-sequentially enabled in $\mathsf{rev}(\rho')$, and the sequential cone $\textsc{SeqCone}(v'_1, v'_2, v'_3, v'_4)$ contains a point $u \in \mathbb{R}^2_{\geq 0}$ with $\|u - t\| \leq D$.

In simple words, instead of proving point 3, we prove that both sequential cones contain a point v which is sufficiently close to t.

Claim 9.9.

The conditions (a) and (a') guarantee that τ is thick.

Indeed, points 1–2 in the definition of thick run are immediate as $P(nM) \leq p(nM)$. For point 3, observe that the inequality (9.1) implies $p(nM)^n \geq \sqrt{2} \cdot D$, which guarantees that the circle $\{u \in \mathbb{R}^2_{\geq 0} \mid \mathsf{dist}(u,t) \leq \sqrt{2} \cdot D\}$ does not touch any half-line ℓ_w induced by a non-negative vector w with $\|w\| \leq p(nM)^n$. In consequence, neither does the square $X := \{u \in \mathbb{R}^2_{\geq 0} \mid \|u - t\| \leq D\}$ inscribed in the circle, and hence X lies between two consecutive half-lines ℓ_w induced by a non-negative vector w with $\|w\| \leq p(nM)^n$. Hence, as $\textsc{SeqCone}(v_1, v_2, v_3, v_4)$ contains some point of X, by lemma 9.3 it includes the whole X, and likewise $\textsc{SeqCone}(v'_1, v'_2, v'_3, v'_4)$. In consequence, the whole X is included in $\textsc{SeqCone}(v_1, v_2, v_3, v_4) \cap \textsc{SeqCone}(v'_1, v'_2, v'_3, v'_4)$ which entails point 3. Claim 9.9 is thus proved.

As condition (a') is fully symmetric to (a), we focus exclusively on proving condition (a), i.e., on constructing sequentially enabled vectors v_1, v_2, v_3, v_4. Vector t lies outside of $p(nM)^n$-belts, hence outside of all the $P(nM)^n$-belts, therefore its norm $\|t\| > P(nM)^n$. Relying on lemma 9.8, let c_1 be the first configuration in the run ρ which enables a semi-positive cycle π_1 of length bounded by $P(nM)$, and let $v_1 = \mathsf{eff}(\pi_1)$. We start with the following obvious claim (let v_0 be some vertical vector, e.g. $v_0 = (0, 1)$):

Claim 9.10.

$\textsc{SeqCone}(v_0)$ contains a point $u \in \mathbb{R}^2_{\geq 0}$ such that $\|u - c_1\| \leq P(nM)^n + nM$.

Indeed, due to lemma 9.8 we may assume $\|c_1\| \leq P(nM)^n + M$ and hence $u = (0, 0)$ satisfies the requirement.

Recall that the relation $\circlearrowright$ defines a total order on pairwise non-collinear non-negative vectors.

Claim 9.11.

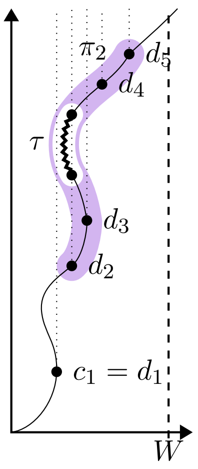
We can assume w.l.o.g. that $v_1 \circlearrowright t$.

Indeed, if v_1 and t were collinear then $t \in \textsc{Cone}(v_1)$ and hence condition (a) would hold.

Split ρ into the prefix ending in c_1 and the remaining suffix: $\rho = \rho_1\,\sigma$, where $\mathsf{trg}(\rho_1) = c_1 = \mathsf{src}(\sigma)$. As the next step we will identify a configuration c_2 in σ which satisfies claim 9.12 (which will serve later as the basis of induction) and enables a cycle π_2 with effect v_2 (as stated in claim 9.13).

Claim 9.12.

$\textsc{SeqCone}(v_0, v_1)$ contains a point $u \in \mathbb{R}^2_{\geq 0}$ such that $\|u - c_2\| \leq P(nM)^n + 2nM$.



The proof of claim 9.12 depends on whether v_1 is positive. If v_1 is so, we simply duplicate the first cycle: $c_2 \coloneqq c_1$ and $\pi_2 \coloneqq \pi_1$, and use claim 9.10. Otherwise, v_1 is vertical due to claim 9.11. If $t[1] \leq W = P(nM)^n + (n+1)M$ then condition (a) holds immediately as $\textsc{SeqCone}(v_1) = \ell_{v_1}$ contains a point $u \in \mathbb{R}^2_{\geq 0}$ with $\|u - t\| \leq P(nM)^n + (n+1)M \leq D$. Therefore, suppose $t[1] > P(nM)^n + (n+1)M$, and define the sequence $d_1, \ldots, d_m$ of configurations as follows: let $d_1 \coloneqq c_1$, and let d_{i+1} be the first configuration in σ with $d_{i+1}[1] > d_i[1]$. Recall that $d_1[1] \leq P(nM)^n + M$, and observe that $d_{i+1}[1] \leq d_i[1] + M$. Thus, by the pigeonhole principle, $m > n$ and hence for some $i < j \leq n+1$ the configurations d_i and d_j must have the same control state. The infix σ_{ij} of the path σ from d_i to d_j is thus a cycle, enabled in d_i, whose effect is positive on the first (horizontal) coordinate. Let $c_2 \coloneqq d_i$. As $c_2[1] \leq P(nM)^n + (n+1)M$, $\textsc{SeqCone}(v_0, v_1) = \ell_{v_0}$ contains necessarily a point $u \in \mathbb{R}^2_{\geq 0}$ such that $\|u - c_2\| \leq P(nM)^n + (n+1)M$, which proves claim 9.12.

Claim 9.13.

The configuration c_2 $\{1\}$-enables a cycle π_2 of length bounded by $p(nM)^n$, such that the first coordinate of $\mathsf{eff}(\pi_2)$ is positive.

Recalling the proof of the previous claim, observe that the first (horizontal) coordinate in the infix σ_{ij} is bounded by $P(nM)^n + (n+1)M$, and think of the second (vertical) coordinate as irrelevant. Let π_2 be the path inducing σ_{ij}. For bounding the length of π_2, as long as π_2 contains a cycle α with vertical effect $(0, w)$, remove α from π_2. This process ends yielding a cycle π_2 of length at most $(P(nM)^n + (n+1)M) \cdot n$, and hence at most $p(nM)^n$ (by the inequality (9.1)), which is $\{1\}$-enabled in c_2, but not necessarily enabled. Let $v_2 := \mathsf{eff}(\pi_2)$.

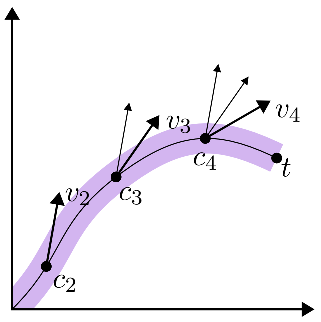
Claim 9.14.

We can assume w.l.o.g. that $v_2 \circlearrowleft t$.

Indeed, if $v_1 = v_2$ then claim 9.11 does the job; otherwise v_1 is vertical and then $t \circlearrowleft v_2$ (or t colinear with v_2) would imply $t \in \textsc{SeqCone}(v_1, v_2)$, hence condition (a) would hold again.

Split σ further into the prefix ending in c_2 and the remaining suffix: $\sigma = \rho_2\sigma'$, where $\mathsf{trg}(\rho_2) = c_2 = \mathsf{src}(\sigma')$. If σ' contains a configuration which $\varnothing$-enables a simple cycle whose effect w belongs to $\measuredangle[t, -v_2)$ then $t \in \textsc{SeqCone}(v_2, w)$ and hence condition (a) holds. We aim at achieving this objective incrementally.

For $i \geq 2$, let c_{i+1} be the first configuration in σ' after c_i that $\varnothing$-enables a simple cycle π_{i+1} with effect $v_{i+1} \in \measuredangle(v_i, -v_i)$. As discussed above, if $v_{i+1} \in \measuredangle[t, -v_i)$ for some i then $t \in \textsc{SeqCone}(v_i, v_{i+1})$ and hence condition (a) holds. Assume therefore that the sequence $v_1, \ldots, v_m$ so defined satisfies $v_{i+1} \in \measuredangle(v_i, t)$ for all $i \geq 2$. Let $c_{m+1} := t$. As vectors $v_3, \ldots, v_m$ are pairwise different, semi-positive and, being effects of simple cycles, have norms at most nM, we know that $m \leq (nM+1)^2 + 1$.



Claim 9.15.

For every $i = 1, \ldots, m$, $\textsc{SeqCone}(v_0, v_i)$ contains a point $u \in \mathbb{R}^2_{\geq 0}$ such that $\|u - c_{i+1}\| \leq P(nM)^n + (i+1)nM$.

Proof of claim 9.15.

By induction on i. The induction base is exactly claim 9.12. For the induction step, we are going to show that $\textsc{SeqCone}(v_0, v_i)$ contains a vector u such that $\|u - c_{i+1}\| \leq P(nM)^n + (i+1)nM$. Decompose the infix of σ' which starts in c_i and ends in c_{i+1} into simple cycles, plus the remaining path $\bar{\rho}$ of length at most n. The norm of the effect $\bar{v}$ of $\bar{\rho}$ is hence bounded by nM, and we have

$$c_{i+1} \;=\; c_i + s + \bar{v},$$

where s is the sum of effects of all the simple cycles. By the definition of v_{i+1}, the effects of all the simple cycles belong to the half-plane $\measuredangle[-v_i, v_i]$, and hence there belongs s. By induction assumption there is $u' \in \textsc{SeqCone}(v_0, v_{i-1})$ such that $\|u' - c_i\| \leq P(nM)^n + inM$. As $v_{i-1} \circlearrowleft v_i$, we also have $u' \in \textsc{SeqCone}(v_0, v_i)$.

Consider the point

$$u \;:=\; u' + s$$

which necessarily belongs to the half-plane $\measuredangle[-v_i, v_i]$ but not necessarily to $\textsf{SeqCone}(v_0, v_i) = \measuredangle[-v_i, v_i] \cap \mathbb{R}^2_{\geq 0}$. Ignoring this issue, by routine calculations we get

$$\|u - c_{i+1}\| \;=\; \|u' + s - c_i - s - \bar{v}\| \;\leq\; \|u' - c_i\| + \|\bar{v}\| \leq \|u' - c_i\| + nM \leq P(nM)^n + (i+1)nM$$

as required for the induction step. Finally, if $u \notin \mathbb{R}^2_{\geq 0}$, translate u towards c_{i+1} until it enters the non-negative quadrant $\mathbb{R}^2_{\geq 0}$; clearly, the translation can only decrease the value of $\|u - c_{i+1}\|$. $\square$

Applying the claim to $i = m$, and knowing that $m \leq (nM+1)^2 + 1$, we get some point $u \in \textsf{SeqCone}(v_0, v_m)$ such that $\|u - t\| \leq P(nM)^n + ((nM+1)^2 + 1) \cdot nM \leq P(nM)^n + (nM+1)^3$. Furthermore, relying on the assumptions that t lies outside all $p(nM)^n$-belts and that $v_1 \circlearrowright t$ we prove, similarly as in the proof of claim 9.9, that $v_1 \circlearrowright u$ and hence the point u belongs also to $\textsf{SeqCone}(v_1, v_m)$. This completes the proof of theorem 9.7. $\square$

9.2.2 Proof of lemma 9.8

Fix a VASS_2 V with n states, and let $M = \|V\|$. We proceed by a sequence of auxiliary lemmas.

Lemma 9.16.

Let ρ be a run such that one of coordinates is smaller than K in all configurations in ρ, and such that $\|\textsf{trg}(\rho)\| > \|\textsf{src}(\rho)\| + KnM$. Then

(I) ρ contains, as an infix, a cycle with vertical or horizontal effect,

(II) ρ contains a configuration enabling such a cycle of length polynomial in KnM.

Proof of lemma 9.16.

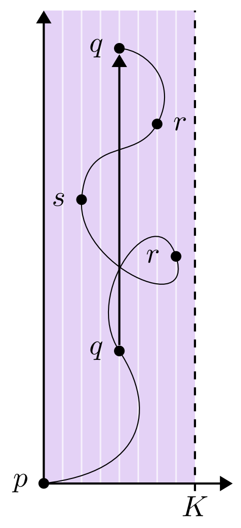
W.l.o.g. assume that the first (horizontal) coordinate is bounded by K in all configurations in ρ. Let $s = \textsf{src}(\rho)$ and $t = \textsf{trg}(\rho)$.

We first prove that ρ contains a cycle with vertical effect. Define a sequence of configurations $c_0, c_1, \ldots, c_m$ as follows. Let c_0 be the first configuration which minimizes the value of the second (vertical) coordinate; clearly $c_0[2] \leq s[2]$. Further, let c_{i+1} be the first configuration in ρ such that $c_{i+1}[2] > c_i[2]$. Thus $c_{i+1}[2] \leq c_i[2] + M$, and in consequence

$$t[2] \leq c_m[2] \leq c_0[2] + mM \leq s[2] + mM.$$

According to the assumption we have $\|t\| > \|s\| + KnM$ hence, as the first coordinate is bounded by K, we deduce the inequality

$$t[2] > s[2] + KnM.$$



The two above inequalities relating $t[2]$ and $s[2]$ imply $m > Kn$. Therefore, there must be two configurations c_i and c_j, for $0 \leq i < j \leq Kn$, with the same control state q and the same first coordinate $c_i[1] = c_j[1]$, and thus the infix ρ_{ij} of ρ from c_i to c_j is a cycle with effect $(0, y)$, where $0 < y \leq (j - i)M \leq KnM$.

Now we bound the length of the cycle. For all configurations in ρ_{ij}, we observe that the first coordinate stays between 0 and $K - 1$, and the second coordinate stays between $c_0[2]$ and $c_j[2]$. We know that $j \leq Kn$, hence $c_j[2] \leq c_0[2] + KnM$. In consequence, the counter values in all configurations in the cycle ρ_{ij} are restricted to at most $K(KnM+1)$ different vectors, and therefore there are at most $L = Kn(KnM+1)$ different configurations in ρ_{ij}. By removing repetitions of configurations, i.e., by removing cycles of effect $(0, 0)$, we reduce the length of the cycle to at most L, which is bounded polynomially in KnM. ☐

Every VASS$_2$ V induces a directed graph whose vertices are control states of V, with an edge from p to q if, and only if, V has a transition of the form (p, v, q). This graph allows us to split control states of V into strongly connected components, which we call briefly SCCs. The following lemma distinguishes two kinds of SCCs:

Lemma 9.17.

Every SCC S satisfies one of the following conditions:

(a) every control state in S belongs to some positive cycle of length polynomial in nM;

(b) the effects of all cycles in S belong to some half-plane containing no positive vector.

Proof of lemma 9.17.

Let U be the set of effects of simple cycles included in S. We consider two cases:

Case 1: $\textsc{Cone}(U)$ contains a positive vector.

Fix an arbitrary positive vector $v \in \textsc{Cone}(U)$. By Caratheodory's Theorem, $v = a_1u_1 + a_2u_2 \in \textsc{Cone}(u_1, u_2)$ for some two vectors $u_1, u_2 \in U$ and $a_1, a_2 \in \mathbb{N}$. By lemma 9.27 we know that $a_1u_1 + a_2u_2$ is positive for some non-negative integers $\alpha_1, \alpha_2 \leq (2M)^2$. We also know that u_1 is the effect of a simple cycle π_1 from, say, state q_1 to q_1; and u_2 is the effect of a simple cycle π_2 from state q_2 to q_2.

Fix a state $q \in S$. As S is strongly connected it contains a cycle π of length at most $3n$ which contains all q, q_1 and q_2. Thus absolute values of $\mathsf{eff}(\pi)$ on both coordinates are at most $3nM$, hence are larger or equal than $-3nM$. Therefore π, together with cycle π_1 repeated $a_1 \cdot (3nM+1)$ times, and with cycle π_2 repeated $a_2 \cdot (3nM+1)$ times, form a cycle with positive effect. The length of this cycle is at most $3n + 2n(2M)^2(3nM + 1)$, hence bounded polynomially in nM. Condition (a) holds.

Case 2: $\textsc{Cone}(U)$ contains no positive vector.

By lemma 9.2 we deduce that $\textsc{Cone}(U)$ is included in some half-plane Π. If Π intersects the positive quadrant $\mathbb{R}^2_{\geq 0}$, rotate the half-plane so that it is disjoint from $\mathbb{R}^2_{\geq 0}$. The so obtained half-plane Π' contains no positive vector and still includes $\textsc{Cone}(U)$, hence condition (b) holds. $\square$

Lemma 9.18.

There is a polynomial Q such that every run ρ within one SCC *with $\|\mathsf{trg}(\rho)\| > Q(nM) \cdot (\|\mathsf{src}(\rho)\| + 1)$ contains a configuration enabling a semi-positive cycle of length at most $Q(nM)$.*

Proof of lemma 9.18.

Let Q_1 and Q_2 be the polynomials from lemma 9.17(II) and lemma 9.16(a), respectively. Let $s = \mathsf{src}(\rho)$ and $t = \mathsf{trg}(\rho)$, and let S be the SCC containg ρ. We split the proof according to the two cases (a) and (b) of lemma 9.17. The proof goes through for every polynomial Q satisfying the following two inequalities:

$$\begin{aligned} Q(x) &\geq Q_1(Q_2(x) \cdot x^2) && \text{(Case 1)} \\ Q(x) &\geq x^2 && \text{(Case 2)} \end{aligned}$$

Case 1: S satisfies (a).

If ρ visits some configuration with both coordinates at least $Q_2(nM) \cdot M = K$ then this configuration necessarily enables a positive cycle of length bounded by $Q_2(nM) \leq Q(nM)$. Otherwise, we know that in every configuration in ρ one of coordinates is smaller than K. W.l.o.g. assume $t[1] < K$. Let ρ' be the longest suffix of ρ such that the first coordinate is bounded by $K - 1$ along ρ', and let $s' = \mathsf{src}(\rho')$. We claim that $\|s'\| \leq \|s\| + K - 1 + M$; indeed, if $s' \neq s$, the first coordinate of the configuration u preceding s' in ρ is at least K, and therefore $u[2] \leq K - 1$, which implies that $s'[2] \leq K - 1 + M$.

By assumption we know that $\|t\| > Q(nM) \cdot (\|s\| + 1)$, and hence necessarily $\|t\| > \|s'\| + KnM$. We can thus apply lemma 9.16(II) to ρ', to learn that some configuration in ρ' enables a vertical cycle of length at most $Q_1(KnM) \leq Q(nM)$.

Case 2: S satisfies (b).

Denoting by U the set of all simple cycles in S, due to condition (b) we know that $\textsc{Cone}(U)$ is included in some half-plane $\Pi = \measuredangle[-w, w]$, where $-w \in \mathbb{N} \times (-\mathbb{N})$ and $w \in (-\mathbb{N}) \times \mathbb{N}$. We aim at showing the following claim:

Claim 9.19.

U contains a vertical or horizontal cycle.

Towards contradiction suppose U contains no vertical nor horizontal cycle. Whenever a vector $p = (-x, y) \in (-\mathbb{N}) \times \mathbb{N}$, for $y > x > 0$, is the effect of a simple cycle, its *ratio* y/x is necessarily bounded by nM. Therefore, the vector w determining Π can be assumed to have ratio bounded by nM as well. Note that all cycles contained as an infix in ρ, necessarily belong to Π. We are going to show bounds on $t[1]$ and $t[2]$ which contradict the assumption on $\|t\|$.

Factor the run ρ into a at most n (not necessarily simple) cycles, interleaved with at most $n-1$ remaining transitions. Thus we have $t = s + r + p$, where $p \in \Pi$ is the total effect of the cycles and r is the total effect of at most $n-1$ transitions. Let p' denote the total effect of those among the cycles whose vertical effect is non-negative (and hence horizontal effect is forcedly negative). Thus

$$t[2] \ \leq \ (s + r + p')[2].$$

As the half-plane $\Pi = \measuredangle[-w, w]$ contains all these cycles, and the ratio of w is bounded by nM as discussed above, we know that ratio of p' is also bounded by nM. In consequence $p'[2] \ \leq \ -p'[1] \ \leq \ (s+r)[1] \cdot nM$, and hence

$$t[2] \ \leq \ (\|s\| + \|r\|) \cdot (1 + nM) \ \leq \ (\|s\| + (n-1)M) \cdot (1 + nM).$$

As the same bound is obtained symmetrically for $t[1]$, we have arrived at a contradiction with the assumption $\|t\| > Q(nM) \cdot (\|s\| + 1)$. Claim 9.19 is thus proved.

Claim 9.20.

The run ρ contains, as an infix, a vertical or horizontal cycle π.

W.l.o.g. supose U contains a vertical cycle. In consequence, no cycle in S has positive first (horizontal) coordinate. Therefore the horizontal coordinate is smaller than $K = s[1] + (n-1)M + 1$ in all configurations in ρ. By lemma 9.16(I) ρ contains, as an infix, a vertical cycle.

Relying on the claim 9.20, w.l.o.g. assume ρ contains a vertical cycle π as infix. For completing the proof of lemma 9.18 we need to bound the length of π. As S satisfies condition (b), it contains no cycle with positive horizontal effect; in consequence, decomposition of π into simple cycles uses only cycles with effect $(0, a)$, where $a \in \mathbb{Z}$. Split these simple cycles into *increasing* $(a > 0)$ and *non-increasing* $(a \leq 0)$. Suppose the length of $\pi = \pi_0$ is greater than n and consider the first simple cycle σ_1 contained as its infix. If σ_1 is non-increasing remove σ_1 from π, thus obtaining the path π_1, and consider the first simple cycle σ_2 contained in π_1 as an infix. Again, remove σ_2 if it is non-increasing. And so on, continue this process until finally certain cycle σ_i in π_{i-1} is increasing. As all the removed simple cycles $\sigma_1, \ldots, \sigma_{i-1}$ were non-increasing, inserting back to π_{i-1} those of them which preceed σ_i necessarily increases the configuration $\mathsf{src}(\sigma_i)$ in π_{i-1} so that it enables σ. The proof is thus completed. $\square$

Proof.

lemma 9.8] Let Q be the polynomial from lemma 9.18. We define a polynomial $P(x) = Q(x) \cdot (x+1)$. Consider a run ρ from a $(0,0)$-configuration to some target configuration t. Let $k \leq n$ be the number of SCCs traversed by the run ρ and, for $i = 1, \ldots, k$, let s_i and

t_i be the first and the last configuration in the i-th SCC, respectively. Then $s_1 = (0, 0)$ and $t_k = t$. Suppose, towards contradiction, that ρ contains no configuration enabling a semi-positive cycle of length at most $P(nM)$. As $Q(nM) \leq P(nM)$, by lemma 9.18 we obtain

$$\|t_i\| \leq Q(n, M) \cdot (\|s_i\| + 1) \tag{9.2}$$

for $i = 1, \ldots, k$. We show by induction on i that $\|t_i\| \leq P(nM)^i$. For $i = 1$ we use (9.2) and the equality $\|s_1\| = 0$, to obtain $\|t_1\| \leq Q(nM) \leq P(nM)$. For the induction step we use (9.2) and the inequality $\|s_{i+1}\| \leq \|t_i\| + M$, to obtain:

$$\begin{aligned}\|t_{i+1}\| \leq Q(nM) \cdot (\|s_{i+1}\| + 1) \leq \\ Q(nM) \cdot (\|t_i\| + M + 1) \leq \\ Q(nM) \cdot (P(nM)^i + M + 1) \leq P(nM)^{i+1},\end{aligned}$$

as required. Thus $\|t\| \leq P(nM)^n$ which contradicts the assumption on $\|t\|$ and therefore completes the proof of lemma 9.8. □

9.3 Dichotomy at work – pumping lemma & short run property

This section illustrates applicability of theorem 9.7. As before, we use symbols n and M for the number of control states, and the norm of a VASS$_2$, respectively. As the first corollary we provide a pumping lemma for VASS$_2$: in case of thin runs apply, essentially, pumping schemes of VASS$_1$, and in case of thick runs use the cycles enabled along a run.

Theorem 9.21. pumping runs of VASS$_2$

There is a polynomial p such that every $(0, 0)$-run τ in a 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 τ_0 and τ_k are also bounded by $p(nM)^n$.

As another application, we derive an alternative proof of the exponential run property for VASS$_2$.

Theorem 9.22. exponential run property

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

We fix from now on a VASS$_2$ $V = (Q, T)$ and the polynomial p of theorem 9.7. Let $A = p(nM)^n$. Both proofs proceed separately for thin and thick runs τ. The polynomials required in theorems 9.21 and 9.22 can be read out from the constructions.

9.3.1 Proof thin

As usual, we use $n = |Q|$ for the number of control states, and $M = \|V\|$ for the norm of V. Assume a $(0,0)$-run τ to be A-thin: every configuration in τ lies in some A-belt $\mathcal{B}_{v,W}$. Fix $W = A + \sqrt{2}M$ and $B = 6WA^2 + 3W$, and let $S = [0,B]^2$. Let $\|v\|_2$ denote the Euclidean norm of v. Note that $\|v\|_2 \leq \sqrt{2}\|v\|$.

Claim 9.23.

The run τ does not change belts outside S, i.e., any two consecutive configurations (q,w), (q',w') in τ satisfying $w, w' \notin S$ share a common belt.

Proof of claim 9.23.

Assume that $w \in \mathcal{B}_{u,A}$ for some u ($\|u\| \leq A$). We will show $w' \in \mathcal{B}_{u,A}$. Notice that $w' \in \mathcal{B}_{u,W}$ (since $\|w' - w\|_2 \leq \sqrt{2}\|w' - w\| \leq \sqrt{2}M$). Towards contradiction assume that v' also belongs to some A-belt $\mathcal{B}_{v,A} \neq \mathcal{B}_{u,A}$ (i.e. v and u non-colinear). Then of course $w' \in \mathcal{B}_{v,W}$ too. We will show that this implies $w' \in S$.

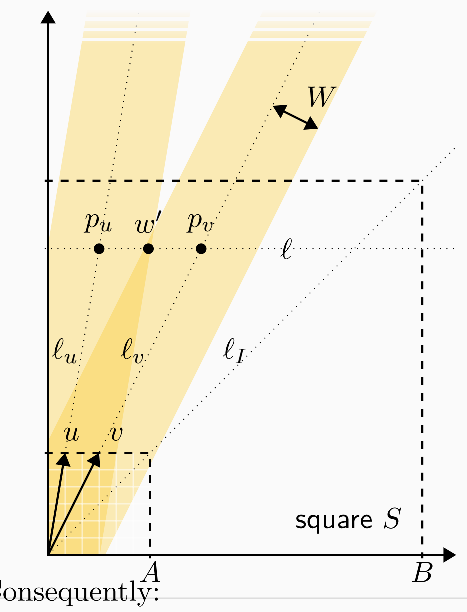
W.l.o.g. assume that $u \circlearrowright v$. Let $I = (1,1)$. Notice that when $u \circlearrowright I \circlearrowright v$ then w' also belongs to $\mathcal{B}_{I,W}$. Thus, we can assume that $u \circlearrowright v \circlearrowright I$ or $I \circlearrowright u \circlearrowright v$. W.l.o.g. let us choose the first option. Note that this implies that $u[2], v[2] > 0$.

Let p_u and p_v be the intersection points of ℓ_u and ℓ_v with the horizontal line $\ell : y = w'[2]$. Their horizontal coordinates are $\frac{u[1]}{u[2]} \cdot w'[2]$ and $\frac{v[1]}{v[2]} \cdot w'[2]$, respectively, so

$$\|p_u - p_v\| = w'[2]\frac{|u[1]v[2] - v[1]u[2]|}{u[2]v[2]}.$$

Because the belts intersect with ℓ at an angle between $45°$ and $90°$, the line segments $\mathcal{B}_{u,W} \cap \ell$ and $\mathcal{B}_{v,W} \cap \ell$ are of length $\leq 2\sqrt{2}W < 3W$. Thus $\|p_u - p_v\| \leq \|p_u - w'\| + \|p_v - w'\| < 6W$. Consequently:

Figure 9.24. Limited intersection



A-belts intersect only within square S.

$$w'[2]\frac{|u[1]v[2] - v[1]u[2]|}{u[2]v[2]} < 6W$$
$$w'[2] < 6W\frac{u[2]v[2]}{|u[1]v[2] - v[1]u[2]|} < 6WA^2 \leq B - 3W < B.$$

Furthermore $p_v[1] < p_v[2] = w'[2]$ and $\|p_v - w'\| < 3W$ so $w'[1] < B$ too, contradiction.

□

Claim 9.25.

Let $C = (A^2 \cdot n)^2$. If τ visits a configuration of norm larger than $D = B + C \cdot A$, then it decomposes into $\tau = \tau_0\,\alpha_1\,\tau_1\,\alpha_2\,\tau_2$, for two cycles α_1, α_2 of opposite effects $\mathsf{eff}(\alpha_1) = -\mathsf{eff}(\alpha_2) \geq (0,0)$ containing jointly at most $C(C+1)$ different configurations.

Proof of claim 9.25.

For a configuration c of norm larger than D, let us decompose τ into

$$\tau = \pi\,\gamma\,\gamma'\,\pi'$$

such that $\mathsf{trg}(\gamma) = c = \mathsf{src}(\gamma')$ and $\gamma\gamma'$ is a maximal infix of π that visits only configurations of norm greater than B. By claim 9.23, there exists unique belt $\mathcal{B} = \mathcal{B}_{u,A}$ that contains $\gamma\gamma'$. Assume w.l.o.g. that $\|u\| > M$. Let us divide $\mathcal{B} \setminus S$ into segments $\mathcal{B}_i$ as follows:

$$S_i \coloneqq [0, B + iu[1]] \times [0, B + iu[2]] \qquad\qquad \mathcal{B}_i \coloneqq \mathcal{B} \cap (S_{i+1} \setminus S_i).$$

Observe that $\gamma\gamma'$ visits more than C initial blocks $\mathcal{B}_i$ starting from $\mathcal{B}_0$ up to $\mathcal{B}_C$, since a single transition cannot 'jump' over a block without visiting it. Let $c_i = (q_i, v_i)$ be the first configuration in γ beloging to $\mathcal{B}_i$, and symmetrically let $c'_i = (q'_i, v'_i)$ be the last configuration in γ' beloging to $\mathcal{B}_i$. Observe that each block $\mathcal{B}_i$ has the same shape as $\mathcal{B}_0$ and differs only by translation by iu. Furthermore, as $\|u\| \leq A$, each $\mathcal{B}_i$ fits inside a square of size A so it contains at most A^2 points. By the pigeonhole principle, there are at least two i, j $(0 \leq i < i + d = j \leq C)$ such that

$$q_i = q_j \qquad v_i + du = v_j \qquad q'_i = q'_j \qquad v'_i + du = v'_j.$$

Taking as α_1 the infix from c_i to c_j, and as α_2 the infix from c'_j to c'_i, we obtain two required cycles. □

Claim 9.26.

Under assumption of claim 9.25, τ decomposes into $\tau = \tau_0\,\alpha_1\,\tau_1\,\alpha_2\,\tau_2$ so that $\tau_0\,\tau_1\,\tau_2$ is also an A-thin $(0,0)$-run.

Proof of claim 9.26.

The same proof as for claim 9.25, with one modification: take as c_i *the last* configuration in γ belonging to $\mathcal{B}_i$, and symmetrically take as c'_i *the first* configuration in γ' belonging to $\mathcal{B}_i$. □

Proof of theorem 9.21.

First part of two—case when τ is A-thin.

Applying claim 9.25 simultaneously to the first belt in which the norm D is exceeded, and to the very last such belt, we get $(0,0)$-runs

$$\tau_0\,\alpha_1^i\,\tau_1\,\alpha_2^i\,\tau_2\alpha_3^i\,\tau_3\,\alpha_4^i\,\tau_4,$$

for $i \in \mathbb{N}$, where cycles α_1, α_2 belong to the first belt and cycles α_3, α_4 belong to the last one. The lengths of the cycles can be reduced to at most $C(C+1)$ by removing repetitions of configurations. Then the length of the very first factor τ_0 can be bounded by $(D+1)^2 + C(C+1)$ by replacing, if needed, cycles α_1, α_2 with the first cycle of effect $(0,0)$ in τ_0. Likewise for the very last factor τ_4. $\square$

Proof of theorem 9.22.

First part of two—case when τ is A-thin.

Immediate using claim 9.26, according to which every A-thin $(0,0)$-run exceeding norm D can be shortened. Once all configurations along a run have norm bounded by D, by eliminating repetitions of configurations we arrive at a run of length at most $n \cdot (D+1)^2$. $\square$

9.3.2 Proof thick

In this section, we assume τ to be A-thick. We rely on the standard tool, cf. [20, Prop. 2] (the norm of a system of inequalities is the largest absolute value of its coefficient, and likewise we define the norm of a solution):

Lemma 9.27.

Let $\mathcal{U}$ be a system of d linear inequalities of norm M with k variables. Then the smallest norm of a non-negative-integer solution of $\mathcal{U}$ is in $\mathcal{O}(k \cdot M)^d$.

Consider a split $\tau = \rho\rho'$, where $\rho = \rho_1\,\rho_2\,\rho_3\,\rho_4\,\rho_5$ and $\rho' = \rho_5'\,\rho_4'\,\rho_3'\,\rho_2'\,\rho_1'$, as well as cycles $\pi_1, \ldots, \pi_4$ and $\pi_1', \ldots, \pi_4'$ given by the definition of thick run. Let $v_1, \ldots, v_4$ and $v_1', \ldots, v_4'$ be the respective effects of $\pi_1, \ldots, \pi_4$ and $\pi_1', \ldots, \pi_4'$. For $j = 1, \ldots, 4$ let $c_j = \mathsf{trg}(\rho_j)$ and for $j = 2, \ldots, 4$ let $e_j \in \mathbb{N}^2$ be the minimal non-negative vector such that the configuration $c_j + e_j$ enables cycle π_j. We define the following system $\mathcal{U}$ of linear inequalities with 6 variables a_1, a_2, a_3, a_4, x, y (max is understood point-wise):

$$a_1v_1 \geq e_2 \tag{9.3}$$

$$a_1v_1 + a_2v_2 \geq \max(e_2, e_3) \tag{9.4}$$

$$a_1v_1 + a_2v_2 + a_3v_3 \geq \max(e_3, e_4) \tag{9.5}$$

$$a_1v_1 + a_2v_2 + a_3v_3 + a_4v_4 = (x, y) \geq e_4 \tag{9.6}$$

(Observe that when $v_1[j] = 0$, i.e., in case when v_1 is vertical or horizontal, $e_j = 0$ and therefore one of the two first inequalities is always satisfied, namely $a_1v_1[j] \geq e_2[j]$.) Likewise, we have a system of inequalities $\mathcal{U}'$ with 6 variables $a_1', a_2', a_3', a_4', x', y'$. Observe that the sequential cone $\textsf{SeqCone}(v_1, v_2, v_3, v_4)$ contains exactly (projections on (x, y) of) non-negative rational solutions of the modified system $\mathcal{U}^{(0,0)}$ obtained by replacing all the right-hand sides with $(0, 0)$. Likewise we define $\mathcal{U}'^{(0,0)}$.

Finally, we define the compound system $\mathcal{C}$ by enhancing the union of $\mathcal{U}$ and $\mathcal{U}'$ with two additional equalities (likewise we define the system $\mathcal{C}^{(0,0)}$)

$$(x, y) = (x', y'). \tag{9.7}$$

Claim 9.28.

$\mathcal{C}$ admits a non-negative integer solution $(a_1, a_2, a_3, a_4, x, y, a_1', a_2', a_3', a_4', x', y')$.

Proof of claim 9.28.

The system $\mathcal{C}^{(0,0)}$ admits a non-negative rational solution as the intersection of the cones $\textsf{SeqCone}(v_1, v_2, v_3, v_4)$ and $\textsf{SeqCone}(v_1', v_2', v_3', v_4')$ is non-empty by assumption. As intersection of cones is stable under multiplications by non-negative rationals, the solution can be scaled up arbitrarily, to yield a non-negative integer one, and even a non-negative integer solution of the stronger system $\mathcal{C}$. $\square$

Claim 9.29.

For every non-negative integer solution of $\mathcal{C}$, for the cycles defined as $\alpha_j := \pi_j^{a_j}$ and $\alpha'_j := (\pi'_j)^{a'_j}$, for $j = 1, 2, 3, 4$, the following path is a $(0, 0)$-run:

$$\rho_1\ \alpha_1\ \rho_2\ \alpha_2\ \rho_3\ \alpha_3\ \rho_4\ \alpha_4\ \rho_5\ \rho'_5\ \alpha'_4\ \rho'_4\ \alpha'_3\ \rho'_3\ \alpha'_2\ \rho'_2\ \alpha'_1\ \rho'_1.$$

Proof of claim 9.29.

The first two inequalities (9.3) enforce that the first cycle π_1 is repeated sufficiently many a_1 times so that π_2 is enabled in configuration $\mathsf{trg}(\rho_1\,\alpha_1\,\rho_2)$. Then the next two inequalities (9.4) enforce that π_1 and π_2 are jointly repeated sufficiently many a_1, a_2 times so that π_2 is still enabled after its last repetition (which guarantees that every of intermediate repetitions of π_2 is also enabled), and that π_3 is enabled in configuration $\mathsf{trg}(\rho_1\,\alpha_1\,\rho_2\,\alpha_2\,\rho_3)$. Likewise for (9.5). Finally, the inequalities (9.6) enforce that $\pi_1, \ldots, \pi_4$ are jointly repeated sufficiently many times so that π_4 is still enabled after its last repetition. Analogous argument, but in the reverse order, applies for the repetitions of $\pi'_4, \ldots, \pi'_1$. Finally, equalities (9.7) ensure that the total effect of $\alpha_1, \ldots, \alpha_4$ is precisely compensated by the total effect of $\mathsf{rev}(\alpha'_1), \ldots, \mathsf{rev}(\alpha'_4)$. □

Proof of theorem 9.21.

Second part of two—case when τ is A-thick.

Consider a solution of $\mathcal{C}$. In particular the sum $\mathsf{eff}(\alpha_1) + \ldots + \mathsf{eff}(\alpha_j)$, as well as $\mathsf{eff}(\mathsf{rev}(\alpha'_1)) + \ldots + \mathsf{eff}(\mathsf{rev}(\alpha'_j))$, is necessarily non-negative for every $j = 1, \ldots, 4$. Therefore, as a direct corollary of claim 9.29, for every $i \in \mathbb{N}$ the path

$$\rho_1\,\alpha_1^i\,\rho_2\,\alpha_2^i\,\rho_3\,\alpha_3^i\,\rho_4\,\alpha_4^i\,\rho_5\,\rho'_5\,(\alpha'_4)^i\,\rho'_4\,(\alpha'_3)^i\,\rho'_3\,(\alpha'_2)^i\,\rho'_2\,(\alpha'_1)^i\,\rho'_1$$

is also a $(0, 0)$-run. For bounding the lengths of cycles we use claim 9.28 and apply lemma 9.27 to $\mathcal{C}$, to deduce that $\mathcal{C}$ admits a non-negative integer solution of norm polynomial in $A = p(nM)^n$. This, together with the bounds on lengths of cycles $\pi_1, \ldots, \pi_4$ and $\pi'_1, \ldots, \pi'_4$ in the definition of A-thick run, entails required bounds on the lengths of the pumpable cycles. Finally, the lengths of the extremal factors ρ_1 and ρ'_1 can be also bounded: if ρ_1 (resp. ρ'_1) is long enough it must admit a repetition of configuration, we add one more cycle determined by the first (resp. last) such repetition, thus increasing k from 8 to 10. □

For proving theorem 9.22 we will need a slightly more elaborate pumping. By the definition of thick run, both coordinates are bounded by A along ρ_1 and ρ'_1. W.l.o.g. assume that no configuration repeats in each of the two runs, and hence their lengths are bounded by A^2.

Let $\mathcal{C}_\delta$ denote the union of of $\mathcal{U}$ and $\mathcal{U}'$ enhanced, this time, by the two equalities

$$(x, y) + (\delta_x, \delta_y) = (x', y').$$

The two additional variables δ_x, δ_y describe, intuitively, possible differences between the total effect of $\pi_1^{a_1}, \ldots, \pi_4^{a_4}$ and the total effect of $\mathsf{rev}(\pi'_1)^{a'_1}, \ldots, \mathsf{rev}(\pi'_4)^{a'_4}$. The projection of any solution of $\mathcal{C}_\delta$ on variables (δ_x, δ_y) we call below a *shift*.

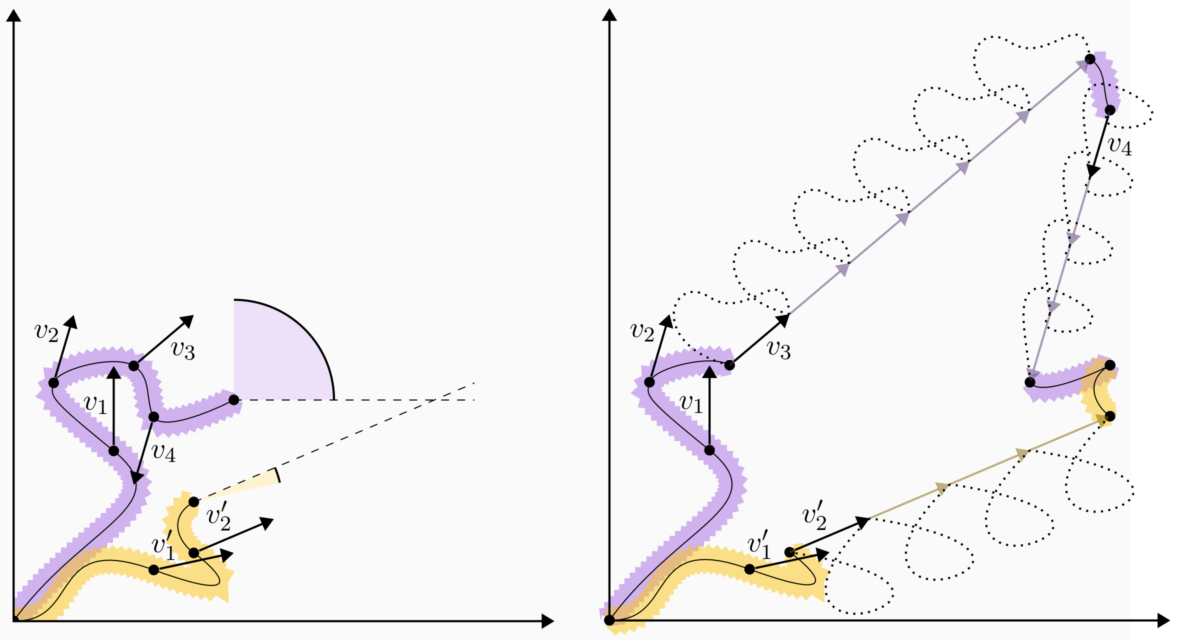
Claim 9.30.

For some non-negative integer m bounded polynomially with respect to A, all the four vectors $(0, m)$, $(m, 0)$, $(0, -m)$ and $(-m, 0)$ are shifts.

Proof of claim 9.30.

We reason analogously as in the proof of claim 9.28, but this time we rely on the assumption that intersection of the cones $\textsc{SeqCone}(v_1, v_2, v_3, v_4)$ and $\textsc{SeqCone}(v'_1, v'_2, v'_3, v'_4)$ is non-trivial, and hence contains, for some $v \in \mathbb{R}^2_{\geq 0}$ and $a \in \mathbb{R}_{\geq 0}$, the points v and $v + (0, a)$. By scaling we obtain an integer point $v' \in \mathbb{N}^2$ and a non-negative integer $m_1 \in \mathbb{N}$ so that v' and $v' + (0, m_1)$ both belong to the intersection of cones. Therefore the vector $(0, m_1)$ is a shift. Likewise we obtain three other non-negative integers $m_2, m_3, m_4 \in \mathbb{N}$ such that $(m_2, 0)$, $(0, -m_3)$ and $(-m_4, 0)$ are all shifts. Each of the integers $m_1, \ldots, m_4$ can be bounded polynomially in A using lemma 9.27. As shifts are stable under multiplication by non-negative integers, it is enough to take as m the least common multiple of the four integers. $\square$

Figure 9.31.



Contracted paths $\widetilde{\rho}, \widetilde{\rho}'$ (left) and reconstructed $(0,0)$-run $\bar{\tau} = \bar{\rho}\,\bar{\rho}'$ (right).

Proof of theorem 9.22.

Second part of two—case when τ is A-thick.

We use m from the last claim to modify all factors of τ except for ρ_1 and ρ'_1, in order to reduce their lengths to at most $n \cdot m^2$. W.l.o.g. assume m to be larger than A (take a sufficient multiplicity of m otherwise); this assumption allows us to proceed uniformly, irrespectively whether v_1 is positive or not. Observe that any path longer than $n \cdot m^2$ must contain two configurations with the same control state whose vectors are coordinate-wise congruent modulo m. As long as this happens, we remove the infix; note that this operation changes the effect of the whole path by a multiplicity of m on every coordinate. If this operation is performed on factors ρ_2, ρ_3, ρ_4, ρ_5, ρ'_5, ρ'_4, ρ'_3, ρ'_2, the paths ρ, ρ' are transformed into contracted paths (see the left picture in figure 9.31) of the form:

$$\widetilde{\rho} \;=\; \rho_1\, \widetilde{\rho}_2\, \widetilde{\rho}_3\, \widetilde{\rho}_4\, \widetilde{\rho}_5, \qquad \widetilde{\rho}' \;=\; \widetilde{\rho}'_5\, \widetilde{\rho}'_4\, \widetilde{\rho}'_3\, \widetilde{\rho}'_2\, \sigma_1,$$

each of total length at most $5n \cdot m^2$. Importantly, their effects $\mathsf{eff}(\widetilde{\rho})$ and $\mathsf{eff}(\widetilde{\rho}')$ are bounded polynomially in A, and their difference is (coordinate-wise) divisible by m:

$$\mathsf{eff}(\widetilde{\rho}) - \mathsf{eff}(\mathsf{rev}(\widetilde{\rho}')) \;=\; (am, bm) \qquad \text{for some integers } a, b \in \mathbb{Z} \text{ polynomial in } A.$$

Our aim is now to pump up the cycles $\pi_1, \ldots, \pi_4$ and $\mathsf{rev}(\pi'_1), \ldots, \mathsf{rev}(\pi'_4)$ (see the right picture in figure 9.31), to finally end up with the paths of the form

$$\bar{\rho} \;=\; \rho_1\, \pi_1^{a_1}\, \widetilde{\rho}_2\, \pi_2^{a_2}\, \widetilde{\rho}_3\, \pi_3^{a_3}\, \widetilde{\rho}_4\, \pi_4^{a_4}\, \widetilde{\rho}_5, \quad \bar{\rho}' \;=\; \widetilde{\rho}'_5\, (\pi'_4)^{a'_4}\, \widetilde{\rho}'_4\, (\pi'_3)^{a'_3}\, \widetilde{\rho}'_3\, (\pi'_2)^{a'_2}\, \widetilde{\rho}'_2\, (\pi'_1)^{a'_1}\, \rho'_1, \tag{9.8}$$

such that $\bar{\tau} = \bar{\rho}\,\bar{\rho}'$ is a $(0,0)$-run. In other words, we aim at $\mathsf{eff}(\bar{\rho}) = \mathsf{eff}(\mathsf{rev}(\bar{\rho}'))$. We are going to use lemma 9.27 twice. For $j = 2, \ldots, 5$ let $c_j := \mathsf{eff}(\rho_1 \widetilde{\rho}_2 \ldots \widetilde{\rho}_j) \in \mathbb{Z}^2$, and let f_j be the minimal non-negative vector such that the configuration $c_{j-1} + f_j$ enables $\widetilde{\rho}_j$. For $j = 2, \ldots, 4$ let $e_j \in \mathbb{N}^2$ be the minimal non-negative vector such that the configuration $c_j + e_j$ enables π_j.® Finally, let e_5 be the minimal non-negative vector such that $c_5 + e_5 \geq (0,0)$. Analogously to the system $\mathcal{U}$ (9.3)–(9.6), we define the system $\widetilde{\mathcal{U}}$ of linear inequalities:

$$\begin{aligned}
a_1 m v_1 &\;\geq\; \max(e_2, f_2)\\
a_1 m v_1 + a_2 m v_2 &\;\geq\; \max(e_2, e_3, f_3)\\
a_1 m v_1 + a_2 m v_2 + a_3 m v_3 &\;\geq\; \max(e_3, e_4, f_4)\\
a_1 m v_1 + a_2 m v_2 + a_3 m v_3 + a_4 m v_4 &\;\geq\; \max(e_4, e_5, f_5)
\end{aligned}$$

In words, $\widetilde{\mathcal{U}}$ requires that every prefix of $\bar{\rho}$ is enabled in the source $(0,0)$-configuration, and that the number of repetitions of every cycle π_i is divisible by m. Clearly $\widetilde{\mathcal{U}}$ has a non-negative integer solution, as v_1 is either positive, or vertical or horizontal in which case v_2 is positive on the relevant coordinate. Likewise we define a system of inequalities $\widetilde{\mathcal{U}}'$ that requires that every prefix of $\mathsf{rev}(\bar{\rho}')$ is enabled in the target $(0,0)$-configuration. Consider some fixed solutions of $\widetilde{\mathcal{U}}$ and $\widetilde{\mathcal{U}}'$ bounded, by the virtue of lemma 9.27, polynomially in A. We have thus two fixed runs $\bar{\rho}$ and $\mathsf{rev}(\bar{\rho}')$ of the form (9.8), with source vector $(0,0)$; the number of repetitions of each cycles is divisible by m, and the difference of their effects is (coordinate-wise) divisible by m:

$$\mathsf{eff}(\bar{\rho}) - \mathsf{eff}(\mathsf{rev}(\bar{\rho}')) \;=\; (am, bm) \qquad \text{for some integers } a, b \in \mathbb{Z} \text{ polynomial in } A.$$

As shifts are closed under addition, by claim 9.30 we know that (am, bm) is a shift. Substituting (am, bm) for (δ_x, δ_y) in the system $\mathcal{C}_\delta$ yields a system which admits, again by lemma 9.27, a solution bounded polynomially in A. We use such a solution to increase the numbers of repetitions of respective cycles $a_1, \ldots, a_4$ and $a'_4, \ldots, a'_1$ in $\bar{\rho}$ and $\bar{\rho}'$, respectively. This turns the path $\bar{\tau} = \bar{\rho}\,\bar{\rho}'$ into a $(0,0)$-run of length bounded polynomially in A. $\square$

Chapter 10

Closing remarks

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 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 DRA- and 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 VASS_2
  So far, the only positive results obtained in this area assume one of the separated languages belongs to VASS_1 (cf. [38]). This is the simplest as yet unsolved case of the general regular separability problem of languages of VASS_k.

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

Bibliography

[1] S. Akshay, Paul Gastin, and Shankara Narayanan Krishna. Analyzing Timed Systems Using Tree Automata. *Logical Methods in Computer Science*, Volume 14, Issue 2, May 2018. URL: `https://lmcs.episciences.org/4489`, `doi:10.23638/LMCS-14(2:8)2018`.

[2] Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126:183–235, 1994.

[3] Rajeev Alur, Limor Fix, and Thomas A. Henzinger. Event-clock automata: a determinizable class of timed automata. *Theor. Comput. Sci.*, 211:253–273, January 1999.

[4] Eugene Asarin and Oded Maler. As soon as possible: Time optimal control for timed automata. In *Proc. of HSCC'99*, HSCC '99, pages 19–30, London, UK, UK, 1999. Springer-Verlag. URL: `http://dl.acm.org/citation.cfm?id=646879.710314`.

[5] Eugene Asarin, Oded Maler, Amir Pnueli, and Joseph Sifakis. Controller synthesis for timed automata. In *Proc. of SSSC'98*, volume 31 of *5th IFAC Conference on System Structure and Control*, pages 447–452, 1998. URL: `http://www.sciencedirect.com/science/article/pii/S1474667017420325`, `doi:https://doi.org/10.1016/S1474-6670(17)42032-5`.

[6] Mohamed Faouzi Atig, Dmitry Chistikov, Piotr Hofman, K. Narayan Kumar, Prakash Saivasan, and Georg Zetzsche. The complexity of regular abstractions of one-counter languages. In *Proc. LICS'16*, pages 207–216, 2016.

[7] Vince Bárány, Christof Löding, and Olivier Serre. Regularity problems for visibly pushdown languages. In *Proc. of STACS'06*, STACS'06, pages 420–431, Berlin, Heidelberg, 2006. Springer-Verlag. URL: `http://dx.doi.org/10.1007/11672142_34`, `doi:10.1007/11672142_34`.

[8] Gerd Behrmann, Alexandre David, Kim G. Larsen, John Hakansson, Paul Petterson, Wang Yi, and Martijn Hendriks. Uppaal 4.0. In *Proceedings of the 3rd International Conference on the Quantitative Evaluation of Systems*, QEST '06, pages 125–126, Washington, DC, USA, 2006. IEEE Computer Society. `doi:10.1109/QEST.2006.59`.

[9] Michael Blondin, Alain Finkel, Stefan Göller, Christoph Haase, and Pierre McKenzie. Reachability in two-dimensional vector addition systems with states is PSPACE-complete. In *Proc. LICS'15*, pages 32–43, 2015.

[10] Mikołaj Bojańczyk. Slightly infinite sets. unpublished. URL: `https://www.mimuw.edu.pl/~bojan/paper/atom-book`.

[11] Mikolaj Bojanczyk, Bartek Klin, and Slawomir Lasota. Automata theory in nominal sets. *Log. Methods Comput. Sci.*, 10(3), 2014. `doi:10.2168/LMCS-10(3:4)2014`.

[12] Mikolaj Bojańczyk and Sławomir Lasota. A machine-independent characterization of timed languages. In *Proc. ICALP 2012*, pages 92–103, 2012.

[13] Patricia Bouyer, Fabrice Chevalier, and Deepak D'Souza. Fault diagnosis using timed automata. In *Proc. of FOSSACS'05*, FOSSACS'05, pages 219–233, Berlin, Heidelberg, 2005. Springer-Verlag. `doi:10.1007/978-3-540-31982-5_14`.

[14] Thomas Brihaye, Thomas A. Henzinger, Vinayak S. Prabhu, and Jean-François Raskin. Minimum-time reachability in timed games. In Lars Arge, Christian Cachin, Tomasz Jurdziński, and Andrzej Tarlecki, editors, *Proc. of ICALP'07*, pages 825–837, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[15] J. Richard Büchi and Lawrence H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969. URL: `http://www.jstor.org/stable/1994916`.

[16] Michaël Cadilhac, Alain Finkel, and Pierre McKenzie. On the expressiveness of Parikh automata and related models. In Rudolf Freund, Markus Holzer, Carlo Mereghetti, Friedrich Otto, and Beatrice Palano, editors, *Proc. of NCMA'11*, volume 282 of *books@ocg.at*, pages 103–119. Austrian Computer Society, 2011.

[17] Franck Cassez, Alexandre David, Emmanuel Fleury, Kim G. Larsen, and Didier Lime. Efficient on-the-fly algorithms for the analysis of timed games. In Martín Abadi and Luca de Alfaro, editors, *Proc. of CONCUR'05*, pages 66–80, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[18] Pierre Chambart and Philippe Schnoebelen. The ordinal recursive complexity of lossy channel systems. In *Proc. LICS 2008*, pages 205–216, 2008.

[19] Dmitry Chistikov, Wojciech Czerwinski, Piotr Hofman, Michal Pilipczuk, and Michael Wehar. Shortest paths in one-counter systems. In *Proc. of FOSSACS'16*, pages 462–478, 2016.

[20] Dmitry Chistikov and Christoph Haase. The taming of the semi-linear set. In *Proc. ICALP'16*, pages 128:1–128:13, 2016.

[21] The 2016 alonzo church award for outstanding contributions to logic and computation. `https://siglog.org/the-2016-alonzo-church-award-for-outstanding-contributions-to-logic-and-computation/`, 2016.

[22] Lorenzo Clemente, Wojciech Czerwiński, Sławomir Lasota, and Charles Paperman. Regular separability of parikh automata. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *Proc. of ICALP'17*, volume 80, pages 117:1–117:13, 2017. URL: `http://drops.dagstuhl.de/opus/volltexte/2017/7497`, `doi:10.4230/LIPIcs.ICALP.2017.117`.

[23] Lorenzo Clemente, Wojciech Czerwiński, Slawomir Lasota, and Charles Paperman. Regular Separability of Parikh Automata. In *The 44th International Colloquium on Automata, Languages, and Programming (ICALP 2017))*, Varsovie, Poland, 2017. URL: `https://hal.archives-ouvertes.fr/hal-01587616`, `doi:10.4230/LIPIcs.ICALP.2017.117`.

[24] Lorenzo Clemente, Wojciech Czerwinski, Slawomir Lasota, and Charles Paperman. Separability of Reachability Sets of Vector Addition Systems. In *Proc. of STACS'17*, volume 66 of *LIPICs*, pages 24:1–24:14, 2017. URL: `http://drops.dagstuhl.de/opus/volltexte/2017/7009`, `doi:10.4230/LIPIcs.STACS.2017.24`.

[25] Lorenzo Clemente, Piotr Hofman, and Patrick Totzke. Timed Basic Parallel Processes. In Wan Fokkink and Rob van Glabbeek, editors, *Proc. of CONCUR'19*, volume 140 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 15:1–15:16, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. URL: `http://drops.dagstuhl.de/opus/volltexte/2019/10917`, `doi:10.4230/LIPIcs.CONCUR.2019.15`.

[26] Lorenzo Clemente, Sławomir Lasota, and Radosław Piórkowski. Determinisability of One-Clock Timed Automata. In Igor Konnov and Laura Kovács, editors, *31st International Conference on Concurrency Theory (CONCUR 2020)*, volume 171 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 42:1–42:17, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. URL: `https://drops.dagstuhl.de/opus/volltexte/2020/12854`, `doi:10.4230/LIPIcs.CONCUR.2020.42`.

[27] Lorenzo Clemente, Sławomir Lasota, and Radosław Piórkowski. Timed games and deterministic separability. *arXiv e-prints*, page arXiv:2004.12868, April 2020. `arXiv:2004.12868`.

[28] Lorenzo Clemente, Sławomir Lasota, and Radosław Piórkowski. Timed games and deterministic separability. In *Proc. of ICALP 2020*, pages 121:1–121:16, 2020.

[29] Lorenzo Clemente, Sławomir Lasota, and Radosław Piórkowski. Timed Games and Deterministic Separability. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, volume 168 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 121:1–121:16, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. URL: `https://drops.dagstuhl.de/opus/volltexte/2020/12528`, `doi:10.4230/LIPIcs.ICALP.2020.121`.

[30] Lorenzo Clemente, Paweł Parys, Sylvain Salvati, and Igor Walukiewicz. The diagonal problem for higher-order recursion schemes is decidable. In *Proc. of LICS'16*, 2016. URL: `http://doi.acm.org/10.1145/2933575.2934527`, `doi:10.1145/2933575.2934527`.

[31] Hubert Comon and Yan Jurski. Timed automata and the theory of real numbers. In *Proc. of CONCUR'99*, CONCUR '99, pages 242–257, London, UK, UK, 1999. Springer-Verlag.

[32] Wojciech Czerwiński and Sławomir Lasota. Regular Separability of One Counter Automata. *Logical Methods in Computer Science*, Volume 15, Issue 2, June 2019. URL: `https://lmcs.episciences.org/5563`.

[33] Wojciech Czerwinski, Slawomir Lasota, Christof Löding, and Radoslaw Piórkowski. New Pumping Technique for 2-Dimensional VASS. In Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen, editors, *44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019)*, volume 138 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 62:1–62:14, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. URL: `http://drops.dagstuhl.de/opus/volltexte/2019/11006`, `doi:10.4230/LIPIcs.MFCS.2019.62`.

[34] Wojciech Czerwinski, Slawomir Lasota, Roland Meyer, Sebastian Muskalla, K. Narayan Kumar, and Prakash Saivasan. Regular Separability of Well-Structured Transition Systems. In Sven Schewe and Lijun Zhang, editors, *29th International Conference on Concurrency Theory (CONCUR 2018)*, volume 118 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 35:1–35:18, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. URL: `http://drops.dagstuhl.de/opus/volltexte/2018/9573`, `doi:10.4230/LIPIcs.CONCUR.2018.35`.

[35] Wojciech Czerwinski, Slawomir Lasota, Roland Meyer, Sebastian Muskalla, K. Narayan Kumar, and Prakash Saivasan. Regular Separability of Well-Structured Transition Systems. In Sven Schewe and Lijun Zhang, editors, *29th International Conference on Concurrency Theory (CONCUR 2018)*, volume 118 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 35:1–35:18, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. URL: `http://drops.dagstuhl.de/opus/volltexte/2018/9573`, `doi:10.4230/LIPIcs.CONCUR.2018.35`.

[36] Wojciech Czerwiński, Wim Martens, and Tomáš Masopust. Efficient separability of regular languages by subsequences and suffixes. In *Proc. of ICALP'14*, ICALP'13, pages 150–161, Berlin, Heidelberg, 2013. Springer-Verlag. URL: `http://dx.doi.org/10.1007/978-3-642-39212-2_16`, `doi:10.1007/978-3-642-39212-2_16`.

[37] Wojciech Czerwiński, Wim Martens, Lorijn van Rooijen, and Marc Zeitoun. A note on decidable separability by piecewise testable languages. In *Proc. of FCT'15*, 2015. URL: `http://dx.doi.org/10.1007/978-3-319-22177-9_14`, `doi:10.1007/978-3-319-22177-9_14`.

[38] Wojciech Czerwiński and Georg Zetzsche. An approach to regular separability in vector addition systems. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '20, page 341–354, New York, NY, USA, 2020. Association for Computing Machinery. `doi:10.1145/3373718.3394776`.

[39] Wojciech Czerwiński and Slawomir Lasota. Regular separability of one counter automata. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12, 2017. `doi:10.1109/LICS.2017.8005079`.

[40] Stéphane Demri and Ranko Lazic. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.*, 10(3):16:1–16:30, 2009. `doi:10.1145/1507244.1507246`.

[41] C. Dima. Computing reachability relations in timed automata. In *Proc. of LICS'02*, pages 177–186, 2002.

[42] Deepak D'Souza and P. Madhusudan. Timed control synthesis for external specifications. In Helmut Alt and Afonso Ferreira, editors, *Proc. of STACS'02*, pages 571–582, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[43] Matthias Englert, Ranko Lazic, and Patrick Totzke. Reachability in two-dimensional unary vector addition systems with states is NL-complete. In *Proc. of LICS '16*, pages 477–484, 2016.

[44] John Fearnley and Marcin Jurdziński. Reachability in two-clock timed automata is PSPACE-complete. *Information and Computation*, 243:26–36, 2015. URL: `http://www.sciencedirect.com/science/article/pii/S0890540114001564`, `doi:http://dx.doi.org/10.1016/j.ic.2014.12.004`.

[45] Diego Figueira, Piotr Hofman, and Slawomir Lasota. Relating timed and register automata. *Math. Struct. Comput. Sci.*, 26(6):993–1021, 2016. `doi:10.1017/S0960129514000322`.

[46] Olivier Finkel. Undecidable problems about timed automata. In *Proc. of FORMATS'06*, FORMATS'06, pages 187–199, Berlin, Heidelberg, 2006. Springer-Verlag. URL: `http://dx.doi.org/10.1007/11867340_14`, `doi:10.1007/11867340_14`.

[47] R. Fraïssé. *Theory of relations*. North-Holland, 1953.

[48] Martin Fränzle, Karin Quaas, Mahsa Shirmohammadi, and James Worrell. Effective definability of the reachability relation in timed automata. *Information Processing Letters*, 153:105871, 2020. URL: `http://www.sciencedirect.com/science/article/pii/S0020019019301541`, `doi:https://doi.org/10.1016/j.ipl.2019.105871`.

[49] Laurent Fribourg. A closed-form evaluation for extended timed automata. Technical report, CNRS & ECOLE NORMALE SUPERIEURE DE CACHAN, 1998.

[50] David Gale and F. M. Stewart. *13. Infinite Games with Perfect Information*, pages 245–266. Princeton University Press, 2016. URL: `https://doi.org/10.1515/9781400881970-014`, `doi:doi:10.1515/9781400881970-014`.

[51] Paul Gastin, Sayan Mukherjee, and B. Srivathsan. Reachability in Timed Automata with Diagonal Constraints. In Sven Schewe and Lijun Zhang, editors, *Proc. of CONCUR'18*, volume 118 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 28:1–28:17, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. URL: `http://drops.dagstuhl.de/opus/volltexte/2018/9566`, `doi:10.4230/LIPIcs.CONCUR.2018.28`.

[52] Paul Gastin, Sayan Mukherjee, and B. Srivathsan. Fast algorithms for handling diagonal constraints in timed automata. In Isil Dillig and Serdar Tasiran, editors,

Computer Aided Verification, pages 41–59, Cham, 2019. Springer International Publishing.

[53] Stefan Göller and Paweł Parys. Bisimulation finiteness of pushdown systems is elementary. In *Proc. of LICS'20*, pages 521–534, 2020.

[54] Jean Goubault-Larrecq and Sylvain Schmitz. Deciding Piecewise Testable Separability for Regular Tree Languages. In *Proc. of ICALP'16*, volume 55 of *LIPIcs*, pages 97:1–97:15, 2016. URL: `http://drops.dagstuhl.de/opus/volltexte/2016/6232`, `doi:10.4230/LIPIcs.ICALP.2016.97`.

[55] R. Govind, Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. Revisiting Local Time Semantics for Networks of Timed Automata. In Wan Fokkink and Rob van Glabbeek, editors, *Proc. of CONCUR 2019*, volume 140 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:15, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. URL: `http://drops.dagstuhl.de/opus/volltexte/2019/10918`, `doi:10.4230/LIPIcs.CONCUR.2019.16`.

[56] Matthew Hague, Jonathan Kochems, and C.-H. Luke Ong. Unboundedness and downward closures of higher-order pushdown automata. In *Proc. of POPL'16*, POPL 2016, pages 151–163, New York, NY, USA, 2016. ACM. URL: `http://doi.acm.org/10.1145/2837614.2837627`, `doi:10.1145/2837614.2837627`.

[57] Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. Better abstractions for timed automata. *Information and Computation*, 251:67–90, 2016. URL: `http://www.sciencedirect.com/science/article/pii/S0890540116300438`, `doi:https://doi.org/10.1016/j.ic.2016.07.004`.

[58] John E. Hopcroft and Jean-Jacques Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theor. Comput. Sci.*, 8:135–159, 1979. `doi:10.1016/0304-3975(79)90041-0`.

[59] H. B. Hunt, III. On the decidability of grammar problems. *J. ACM*, 29(2):429–447, April 1982. URL: `http://doi.acm.org/10.1145/322307.322317`, `doi:10.1145/322307.322317`.

[60] Marcin Jurdziński and Ashutosh Trivedi. Reachability-time games on timed automata. In *Proc. of ICALP'07*, pages 838–849, Berlin, Heidelberg, 2007. Springer-Verlag. URL: `http://dl.acm.org/citation.cfm?id=2394539.2394637`.

[61] Michael Kaminski and Nissim Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994.

[62] Richard M. Karp and Raymond E. Miller. Parallel program schemata. *J. Comput. Syst. Sci.*, 3(2):147–195, 1969.

[63] Ayrat Khalimov, Benedikt Maderbacher, and Roderick Bloem. Bounded synthesis of register transducers. In Shuvendu Lahiri and Chao Wang, editors, *Automated Technology for Verification and Analysis*, Lecture Notes in Computer Science, pages 494–510, 2018. 16th International Symposium on Automated Technology for Verification and Analysis, ATVA 2018 ; Conference date: 07-10-2018 Through 10-10-2018. `doi:10.1007/978-3-030-01090-4_29`.

[64] Bartek Klin, Sławomir Lasota, and Szymon Toruńczyk. Nondeterministic and co-nondeterministic implies deterministic, for data languages. *Proc. of FOSSACS'21*, 12650:365–384, 03 2021. URL: `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7984108/`, `doi:10.1007/978-3-030-71995-1{\_}19`.

[65] Eryk Kopczynski. Invisible pushdown languages. In *Proc. of LICS'16*, pages 867–872, 2016. URL: `http://doi.acm.org/10.1145/2933575.2933579`, `doi:10.1145/2933575.2933579`.

[66] S. Rao Kosaraju. Decidability of reachability in vector addition systems (preliminary version). In *STOC'82*, pages 267–281, 1982.

[67] Pavel Krčál and Radek Pelánek. On sampled semantics of timed systems. In Sundar Sarukkai and Sandeep Sen, editors, *Proc. of FSTTCS'05*, volume 3821 of *LNCS*, pages 310–321. Springer, 2005. URL: `http://dx.doi.org/10.1007/11590156_25`.

[68] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. of CAV'11*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

[69] Slawomir Lasota. Decidability border for petri nets with data: WQO dichotomy conjecture. In Fabrice Kordon and Daniel Moldt, editors, *Proc. PETRI NETS 2016*, volume 9698 of *Lecture Notes in Computer Science*, pages 20–36. Springer, 2016. `doi:10.1007/978-3-319-39086-4\_3`.

[70] Slawomir Lasota and Igor Walukiewicz. Alternating timed automata. *ACM Trans. Comput. Logic*, 9(2):10:1–10:27, 2008. URL: `http://doi.acm.org/10.1145/1342991.1342994`, `doi:10.1145/1342991.1342994`.

[71] Michel Latteux. Langages à un compteur. *J. Comput. Syst. Sci.*, 26(1):14–33, 1983.

[72] Oded Maler and Amir Pnueli. On recognizable timed languages. In Igor Walukiewicz, editor, *Proc. of FOSSACS'04*, volume 2987 of *LNCS*, pages 348–362. Springer Berlin Heidelberg, 2004. URL: `http://dx.doi.org/10.1007/978-3-540-24727-2_25`, `doi:10.1007/978-3-540-24727-2_25`.

[73] Ernst W. Mayr. An algorithm for the general petri net reachability problem. In *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*, STOC '81, page 238–246, New York, NY, USA, 1981. Association for Computing Machinery. `doi:10.1145/800076.802477`.

[74] Ernst W. Mayr. An algorithm for the general Petri net reachability problem. In *STOC'81*, pages 238–246, 1981.

[75] Richard Mayr. Undecidable problems in unreliable computations. *Theor. Comput. Sci.*, 297(1-3):337–354, March 2003. URL: `http://dx.doi.org/10.1016/S0304-3975(02)00646-1`, `doi:10.1016/S0304-3975(02)00646-1`.

[76] Joshua Moerman, Matteo Sammartino, Alexandra Silva, Bartek Klin, and Michał Szynwelski. Learning nominal automata. In *Proc. of POPL'17*, POPL 2017, pages 613–625, New York, NY, USA, 2017. ACM. URL: `http://doi.acm.org/10.1145/3009837.3009879`, `doi:10.1145/3009837.3009879`.

[77] Frank Neven, Thomas Schwentick, and Victor Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Logic*, 5(3):403–435, July 2004. `doi:10.1145/1013560.1013562`.

[78] Brian Nielsen and Arne Skou. Automated test generation from timed automata. *International Journal on Software Tools for Technology Transfer*, 5(1):59–77, Nov 2003. `doi:10.1007/s10009-002-0094-1`.

[79] Joël Ouaknine and James Worrell. On the language inclusion problem for timed automata: Closing a decidability gap. In *19th IEEE Symposium on Logic in Computer Science (LICS 2004), 14-17 July 2004, Turku, Finland, Proceedings*, pages 54–63, 2004. `doi:10.1109/LICS.2004.1319600`.

[80] Joël Ouaknine and James Worrell. On the decidability and complexity of Metric Temporal Logic over finite words. *Logical Methods in Computer Science*, Volume 3, Issue 1, February 2007. URL: `https://lmcs.episciences.org/2230`, `doi:10.2168/LMCS-3(1:8)2007`.

[81] Thomas Place, Lorijn Rooijen, and Marc Zeitoun. Separating regular languages by piecewise testable and unambiguous languages. In Krishnendu Chatterjee and Jirí Sgall, editors, *Proc. of MFCS'13*, pages 729–740. Springer, 2013. URL: `http://dx.doi.org/10.1007/978-3-642-40313-2_64`, `doi:10.1007/978-3-642-40313-2_64`.

[82] Thomas Place, Lorijn van Rooijen, and Marc Zeitoun. Separating regular languages by locally testable and locally threshold testable languages. *LMCS*, 10(3), 2014. URL: `http://dx.doi.org/10.2168/LMCS-10(3:24)2014`, `doi:10.2168/LMCS-10(3:24)2014`.

[83] Thomas Place and Marc Zeitoun. Going higher in the first-order quantifier alternation hierarchy on words. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Proc. of ICALP'14*, pages 342–353, Berlin, Heidelberg, 2014. Springer. URL: `http://dx.doi.org/10.1007/978-3-662-43951-7_29`, `doi:10.1007/978-3-662-43951-7_29`.

[84] Thomas Place and Marc Zeitoun. Separating regular languages with first-order logic. *Logical Methods in Computer Science*, 12(1), 2016. URL: `http://dx.doi.org/10.2168/LMCS-12(1:5)2016`, `doi:10.2168/LMCS-12(1:5)2016`.

[85] Michael O. Rabin. Weakly definable relations and special automata. In Yehoshua Bar-Hillel, editor, *Mathematical Logic and Foundations of Set Theory*, volume 59 of *Studies in Logic and the Foundations of Mathematics*, pages 1 – 23. Elsevier, 1970. URL: `http://www.sciencedirect.com/science/article/pii/S0049237X08719293`, `doi:https://doi.org/10.1016/S0049-237X(08)71929-3`.

[86] H. G. Rice. Classes of recursively enumerable sets and their decision problems. *Transactions of the American Mathematical Society*, 74(2):358–358, February 1953. `doi:10.1090/s0002-9947-1953-0053041-6`.

[87] J. Richard Büchi. Symposium on decision problems: On a decision method in restricted second order arithmetic. In Ernest Nagel, Patrick Suppes, and Alfred Tarski,

editors, *Logic, Methodology and Philosophy of Science*, volume 44 of *Studies in Logic and the Foundations of Mathematics*, pages 1–11. Elsevier, 1966. URL: `https://www.sciencedirect.com/science/article/pii/S0049237X09705646`, `doi:https://doi.org/10.1016/S0049-237X(09)70564-6`.

[88] Jeffrey Shallit. *A Second Course in Formal Languages and Automata Theory*. Cambridge University Press, 2008.

[89] P. Vijay Suman, Paritosh K. Pandya, Shankara Narayanan Krishna, and Lakshmi Manasa. Timed automata with integer resets: Language inclusion and expressiveness. In *Proc. of FORMATS'08*, FORMATS '08, pages 78—92, Berlin, Heidelberg, 2008. Springer-Verlag. `doi:10.1007/978-3-540-85778-5_7`.

[90] Thomas G. Szymanski and John H. Williams. Noncanonical extensions of bottom-up parsing techniques. *SIAM Journal on Computing*, 5(2):231–250, 1976. URL: `http://dx.doi.org/10.1137/0205019`, `arXiv:http://dx.doi.org/10.1137/0205019`, `doi:10.1137/0205019`.

[91] Martin Tappler, Bernhard K. Aichernig, Kim Guldstrand Larsen, and Florian Lorber. Time to learn - learning timed automata from tests. In Étienne André and Mariëlle Stoelinga, editors, *Proc. of FORMATS'19*, pages 216–235, Cham, 2019. Springer International Publishing.

[92] Ramanathan S. Thinniyam and Georg Zetzsche. Regular Separability and Intersection Emptiness Are Independent Problems. In Arkadev Chattopadhyay and Paul Gastin, editors, *Proc. of FSTTCS'19*, volume 150 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 51:1–51:15, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. URL: `https://drops.dagstuhl.de/opus/volltexte/2019/11613`, `doi:10.4230/LIPIcs.FSTTCS.2019.51`.

[93] Stavros Tripakis. Folk theorems on the determinization and minimization of timed automata. *Inf. Process. Lett.*, 99(6):222–226, September 2006.

[94] Leslie G. Valiant. Regularity and related problems for deterministic pushdown automata. *J. ACM*, 22(1):1–10, January 1975. URL: `http://doi.acm.org/10.1145/321864.321865`, `doi:10.1145/321864.321865`.

[95] Rüdiger Valk and Guy Vidal-Naquet. Petri nets and regular languages. *Journal of Computer and System Sciences*, 23(3):299–325, 1981. URL: `http://www.sciencedirect.com/science/article/pii/0022000081900672`, `doi:http://dx.doi.org/10.1016/0022-0000(81)90067-2`.

[96] Sicco Verwer, Mathijs de Weerdt, and Cees Witteveen. An algorithm for learning real-time automata. In *Proc of. the Annual Belgian-Dutch Machine Learning Conference (Benelearn'078)*, 2007.