UNIVERSITÀ DEGLI STUDI DI UDINE
HIC SUNT FUTURA

DEPARTMENT OF MATHEMATICS, COMPUTER SCIENCE AND PHYSICS

MASTER THESIS IN
COMPUTER SCIENCE

Hybrid Modeling, Simulation, and Sub-Optimal Synthesis of a controller for the Ultrafiltration of Industrial Wastewater

CANDIDATE
Francesco Zuccato

SUPERVISOR
Prof. Carla Piazza

CO-SUPERVISORS
Prof. Giorgio Bacci
Prof. Vittorio Boffa

Academic Year 2023-2024

Institute Contacts
Dipartimento di Scienze Matematiche, Informatiche e Fisiche
Università degli Studi di Udine
Via delle Scienze, 206
33100 Udine — Italia
+39 0432 558400
https://www.dmif.uniud.it/

Acknowledgements

I would like to express my deepest gratitude to my professor, supervisor, and mentor, Prof. Carla Piazza. Throughout both my Bachelor's and Master's studies, she has been an pillar of support and guidance. During my Bachelor's degree, she offered me a internship opportunity in the Computational Biology and Bioinformatics Lab that she coordinates. In my Master's program, she assisted me in securing an international internship, culminating in a positive response from Prof. Giorgio Bacci.

This thesis would not have been possible without the support and collaboration of Prof. Giorgio Bacci, Associate Professor in Computer Science, and Prof. Vittorio Boffa, Associate Professor of Inorganic Chemistry. They provided me with the opportunity to intern at the University of Aalborg, Denmark, where I spent three months working on the project described in this thesis. Their continuous suggestions, feedback, and reviews were instrumental in shaping this work.

I am also profoundly grateful to my entire family. My parents, Catia and Giuseppe, and my elder siblings, Jessica and Davide, along with their spouses, Luca and Silvia, have been my constant support system throughout all of my life. I also want to express my love to my nieces and nephews, Ester, Anna, Benedetta, and Giovanni, who bring immense joy to my life. To my beloved sister Deborah, joking with you will always be my favorite pastime. A special thank you to Silvana, who will always be like a second mother to me.

I want to thank all my friends who have shared some of the best moments of my life and always know how to make me smile: Max, Seba, Due, Moro, Erik, Pelle, Ornella, Irene, Cerve, Bru.

Last but not least, I want to acknowledge any names I may have missed, including my football teammates and university colleagues.

Ringraziamenti

Vorrei esprimere la mia più profonda gratitudine alla mia professoressa, relatrice e mentore, la Prof.ssa Carla Piazza. Durante i miei studi sia triennali che magistrali, è stata un pilastro di supporto e guida. Durante la laurea triennale mi ha offerto un'opportunità di tirocinio nel laboratorio di Biologia Computazionale e Bioinformatica da lei coordinato. Durante la magistrale, mi ha assistito nell'ottenimento di un tirocinio internazionale, culminato con una risposta positiva da parte del Prof. Giorgio Bacci.

Questa tesi non sarebbe stata possibile senza il supporto e la collaborazione del Prof. Giorgio Bacci, Professore Associato di Informatica, e del Prof. Vittorio Boffa, Professore Associato di Chimica Inorganica. Mi hanno dato l'opportunità di fare un tirocinio presso l'Università di Aalborg, in Danimarca, dove ho trascorso tre mesi lavorando al progetto descritto in questa tesi. I loro continui suggerimenti, feedback e revisioni sono stati determinanti nel dare forma a questo lavoro.

Voglio ringraziare tutta la mia famiglia. I miei genitori, Catia e Giuseppe, i miei fratelli maggiori, Jessica e Davide, che mi sono sempre stati affianco ed hanno rappresentato un punto d'appoggio e di riferimento per tutta la mia vita. Voglio esprimere il mio affetto ai miei cognati Luca e Silvia e ai miei nipoti, Ester, Anna, Benedetta e Giovanni, è sempre bello passare del tempo con voi. Alla mia cara sorella Deborah (Debby solo per me), scherzare con te (di te) sarà sempre il mio passatempo preferito. Un ringraziamento speciale a Silvana, che per me sarà sempre come una seconda mamma.

Voglio ringraziare tutti i miei amici, con cui ho condiviso alcuni dei momenti più belli della mia vita e sanno sempre come farmi sorridere: i "bibionesi" Max (pocooossss), Seba "a Ferragosto non combino", Due (minuti e arrivo) e gli zoppolani Moro "********* la ******** destra", Erik (La Pianta) "Hai tutto?", Pelle "L'ultima e poi andiamo", Ornella (La Fighetta), Ire "daiiii amiiiiooooooo", (Papà) Cervo, Bru "Se vuoi ti passo a prendere".

Ultimi ma non meno importanti, voglio ricordare tutti i nomi che potrei aver perso, compresi i miei compagni di calcio ed i miei colleghi universitari.

Abstract

Historically, many industries have required large amounts of fresh water for their processes. The typical water cycle involves buying fresh water, using it in the process, and disposing of the wastewater without recovery, necessitating a constant supply of new water. This pattern is known as "*take-make-dispose*" and has always been the standard, according to the traditional economy. But nowadays, **water has become a scarce resource**.

The objective of this thesis is to exploit and adapt solid formal methods and tools developed in the area of Computer Science with the aim of reducing the water footprint of industries, helping them in the transition **towards a circular economy**. Ultrafiltration (UF) is an industrial process used to purify liquids by filtering them through specialized membranes. In particular, UF is a crucial technology for pretreating wastewater, enabling its reintroduction into the industrial process. With UF, is possible to filter a large volume of wastewater, producing a small amount of solid waste and a significant amount of reusable clean water.

In this thesis, we first examined formal tools for representing complex real systems, such as Timed Automata (TA) and **Hybrid Automata** (HA). Unfortunately, many problems with HA are well-known to be undecidable. To analyze and verify properties of HA, we employed Statistical Model Checking (SMC). SMC is a Monte Carlo-based method that allows to estimate system's properties without requiring a complete formal analysis.

Then, we develop a computational model to study, simulate, and **optimize an ultrafiltration process**. The model relies on Hybrid Automata, a formal object designed to describe complex systems that exhibit both continuous and discrete behaviors in their dynamics. The model is composed of two main components: one is the environment, in which ultrafiltration is simulated using ordinary differential equations, and the other is the controller, which can alter the working conditions.

The hybrid automaton is implemented in **Uppaal**, a software tool for modeling, simulating, and verifying real-time systems. Uppaal Stratego enables the **synthesis of sub-optimal controllers** based on a given cost function. Utilizing SMC, Stratego explores a finite number of possible executions (finite *traces*) to minimize a cost function, which in ultrafiltration is primarily energy consumption.

Finally, we present the experimental results, demonstrating that the model's simulated data closely estimates real data. Additionally, we show how various strategies can be defined and optimized, such as minimizing total energy consumption for a cost-effective approach or minimizing total time for a faster solution.

Sommario

Storicamente, le industrie hanno sempre richiesto grandi quantità di acqua pulita per i loro processi. Il tipico ciclo dell'acqua prevede l'acquisto di acqua pulita, il suo utilizzo nel processo e lo smaltimento delle acque reflue senza recupero, rendendo necessario un rifornimento costante di nuova acqua. Questo schema è noto come "*take-make-dispose*" ed è sempre stato lo standard, secondo l'economia tradizionale. Ma oggigiorno l'**acqua è diventata una risorsa scarsa**.

L'obiettivo di questa tesi è sfruttare e adattare metodi e strumenti formali sviluppati nel settore dell'informatica con l'obiettivo di ridurre l'impronta idrica delle industrie, aiutandole nella transizione **verso un'economia circolare**. L'ultrafiltrazione (UF) è un processo industriale utilizzato per purificare i liquidi filtrandoli attraverso membrane specializzate. In particolare, l'UF è una tecnologia cruciale per il pretrattamento delle acque reflue, consentendone la reintroduzione nel processo industriale. Con l'UF è possibile filtrare un grande volume di acque reflue, producendo una piccola quantità di rifiuti solidi e una notevole quantità di acqua pulita riutilizzabile.

In questa tesi, abbiamo prima esaminato gli strumenti formali per rappresentare complessi sistemi in tempo reale, come i gli automi temporizzati (TA) e gli automi ibridi (HA). Sfortunatamente, è noto che molti problemi legati agli HA sono indecidibili. Perciò, per analizzare e verificare le proprietà dell'HA, abbiamo utilizzato il model checking statistico (SMC). Lo SMC è un metodo basato sui metodi Monte Carlo che consente di stimare le proprietà del sistema senza richiedere un'analisi formale completa.

Successivamente, abbiamo sviluppato un modello computazionale per studiare, simulare e **ottimizzare un processo di ultrafiltrazione**. Il modello si basa su un automa ibrido, un oggetto formale progettato per descrivere sistemi complessi che mostrano comportamenti sia continui che discreti nelle loro dinamiche. Il modello è composto da due componenti principali: uno è l'ambiente, in cui l'ultrafiltrazione viene simulata utilizzando le equazioni differenziali ordinarie, e l'altro è il controllore, che può alterare alcuni parametri.

L'automa ibrido è stato implementato in **Uppaal**, uno strumento software che serve per modellare, simulare e verificare i sistemi in tempo reale. Uppaal Stratego consente la **sintesi di controllori subottimali** in base ad una data funzione di costo. Utilizzando lo SMC, Stratego esplora un numero finito di possibili esecuzioni col fine di minimizzare una funzione di costo, che nel caso dell'ultrafiltrazione è principalmente il consumo di energia.

Infine, presentiamo i risultati sperimentali, dimostrando che i dati simulati del modello stimano molto fedelmente i dati reali. Inoltre, mostriamo come è possibile definire e ottimizzare varie strategie, come ridurre al minimo il consumo energetico totale per un approccio economicamente vantaggioso o ridurre al minimo il tempo totale per una soluzione più rapida.

Contents

List of Figures

1

Introduction

Ultrafiltration (UF) essentially consists of filtering a liquid by pushing it against a membrane with tiny pores, which acts as a barrier and retains suspended solids, bacteria, proteins, etc. Ultrafiltration is nowadays widely used across many sectors. In the food and beverage industry, it is employed for clarifying juices, wines, and beer. In the pharmaceutical sector, UF purifies water by removing bacteria. In the textile industry, it removes dyes and particles from wastewater, enabling the reuse of dye baths.

UF has also become a key technology for **pretreating wastewater** and making it suitable for reuse. Industries that demand large volumes of water can benefit from UF in two significant ways: on one side it reduces the high disposal costs of large volumes of wastewater and, on the other side, it enables the reuse of that same water. Moreover, according to [40], by 2030 the daily demand of water will exceed more than 69% of the total amount of accessible water. Thus, by reintroducing water into production, industries are more sustainable, and lower their **water footprint**. Finally, considering the upcoming scarcity of fresh water, UF will soon be not just a cost-saving measure but a necessity to keep factories operational.

In this thesis, we first studied possible formal tools that we could use to represent our model. Together with the latter come formal languages, i.e. stochastic and temporal logics, that allow to ask questions and verify properties of the given models. This part corresponds to section two, which is quite extensive because most of the models and techniques presented here was not part of my studies.

Then, we present the implemented **computational model**, able to forecast the flux of an ultrafiltration process, accounting for fouling and variations in the operating pressure. Formally, we represented it by using a **hybrid automaton** [31] made of discrete variables and of continuous variables. The dynamics of continuous variables are defined via ordinary differential equations (ODEs), while discrete variables -which can vary only at discrete time steps- by ordinary state updates. In a hybrid automaton, inside each location continuous variables evolve. At fixed intervals, the evolution is stopped and the **controller** (which emulates a human operator) has the possibility to alter the working conditions, i.e., the operating pressure. By aiming to optimize a controller, the original hybrid automaton can be seen as an **hybrid game**, where the controller is opposed to the environment.

The model has been successfully implemented in Uppaal [39], a tool for the verification of real-time systems, jointly developed by the universities of Uppsala (Sweden) and Aalborg (Denmark). Recently, inside Uppaal has been introduced Stratego, an extension that allows to **synthesize statistical strate-**

gies, rooted in **statistical model checking (SMC)** [24]. SMC is an efficient simulation-driven method that supports the estimation and the inference of quantitative properties on complex observable systems.

The main achievements of this thesis are (i) a satisfactory modelization of the real system, (ii) the possibility of learning strategies on such a complex system, and (iii) a straightforward way to adapt the model to custom requirements (by changing some parameters). A particular contribution regards point (ii), where we exploit **Uppaal Stratego** [23] potentiality, by discovering cost-(sub)optimal strategies that guarantee fast and cheap filtration processes even for complex systems modeled as hybrid automata [22].

The work can be summarized in five steps. The **first** one is the experimental part, where we used the pilot-plant, by ultrafiltrating some wastewater samples, made of a mixture of tap water and ink. We then gathered the data collected from the sensors of the plant. The **second** part aims to construct the model. In practice, it consists of data analysis, including making and verifying assumptions, estimating parameters, and also forecasting the real data further in the future. The **third** one dwells in the Uppaal implementation of the hypothesized model as a hybrid automaton. In this part, we carefully chose which variables could be discretized and which stayed continuous. We defined the locations and the transitions of the automaton, alongside parameters, constants, discrete variables, and continuous variables. The **fourth** part accounts for validating the model. We compared the real data with Uppaal's simulations that reproduce the same working conditions of the experiment. The **last** chunk is the driving spirit of this project: integrating the model of ultrafiltration with statistical model checking and Uppaal Stratego. Using temporal logic, one can query how the model will behave over time, verify properties on it, and synthesize control strategies. For example, it is possible to estimate the minimum time that will be required to filtrate a given amount of wastewater, the average energy that will be required, and, most importantly, to identify the appropriate tradeoff between time and costs.

1.1 Motivations

The circular economy is a sustainable economic model that emphasizes the continual use of resources, minimizing waste, and promoting environmental conservation. This approach contrasts with the traditional linear economy, which follows a "*take-make-dispose*" pattern. In a circular economy, products and materials are designed to be reused, repaired, and recycled, thereby extending their lifecycle and reducing the strain on natural resources.

Reusing industrial wastewater is a crucial component of the circular economy. Industrial processes consume vast amounts of water and generate significant quantities of wastewater, which can contain harmful pollutants. Treating and reusing this wastewater not only conserves freshwater resources but also reduces environmental pollution. By reintegrating treated wastewater back into industrial processes or other applications, industries can lower their water footprint, enhance resource efficiency, and contribute to a more sustainable and resilient economic system. This practice not only supports environmental sustainability but also offers economic benefits by reducing costs of both freshwater procurement and wastewater disposal.

1.2 AI for the People: Project Overview

The **AI for the People Center** [46] was initiated in 2019 to coordinate the diverse AI activities across all four faculties at Aalborg University: the Technical Faculty of IT and Design, the Faculty of Engineering and Science, the Faculty of Humanities and Social Sciences, and the Faculty of Medicine. Its name highlights the commitment to developing artificial intelligence (AI) solutions that will impact people's lives. The center's multidisciplinary approach is crucial for examining the ethical and legal implications that AI technologies pose to our society.

The project presented in this thesis is one among the multitude of projects part of the AI for the People Center. The project is funded by Aalborg University, and aims to optimize process parameters such as transmembrane pressure and permeate flow to minimize energy consumption costs and reduce waste production. An ultrafiltration process can be modeled using a continuous dynamical system with (non-linear) differential equations describing the feed and permeate over time. Some process parameters can be expressed as the synthesis of a control problem, i.e., a system that interacts with the filtration process to optimally tune the parameters over time.

However, the theory of non-linear control synthesis has severe limitations due to the undecidability of many problems for the hybrid systems. To overcome this issue, Uppaal-SMC (Statistical Model Checking) was utilized, allowing properties to be verified based on simulations of a model rather than a complete exploration of the state space. To minimize energy costs, Uppaal Stratego was employed, which, given a cost function and an automaton, generates large amounts of simulations to identify sub-optimal cost strategies.

1.3 Thesis Outline

The thesis consists of eleven chapters. The **current** chapter provides context on the research, in particular, introduces what ultrafiltration is, why it matters, and how such a process can be optimized using formal methods techniques. Chapter **two** contains all the necessary formal background to understand what a hybrid automaton is and how to combine it with statistical model checking and controller synthesis. Chapter **three** provides some basic knowledge on separation processes in general, what types exist, in what they differ, etc. Moreover, it lists the different properties that different membranes have, and also explains the main factors present in a filtration process (pressure, temperature, flow, viscosity, etc.) Chapter **four** deepens specifically into ultrafiltration: it explains its main challenges and then provides a list of the most important formulas that model the flux. Chapter **five** details the methodology we used to conduct the experiments: the characteristics of the plant, the data analysis, and the estimation of the parameters. Chapter **six** finally contains our developed mathematical model. It details why we decided that a given variable x should be assumed to be discrete rather than continuous (or vice-versa). Most importantly, to each variable, we attach its (differential) equation that defines it plus some comments. Chapter **seven** briefly explores the potential that Uppaal offers, how it works, and its main components. Chapter **eight** describes the implementation of the data analysis in Python and of the model in Uppaal. Chapter **nine** contains some experimental results, such as the evaluation of the model, and gives a hint of the uncountable questions that can be statistically verified on the model. Finally, the **last two** chapters contain our conclusions on the research and the possible future next steps of this project.

2

Background and Theoretical Framework

Formal methods aim to provide systematic, mathematically rigorous techniques for the specification, design, and verification of complex systems. These methods emerged as a response to the growing need for trustworthy systems, where informal techniques alone were insufficient to ensure correctness, safety, and reliability. In the modeling process of the systems and their behaviors logic-based languages and tools are used. Formal tools offer many advantages, which are:

- **Precision**: Precise specification of the system is required, leaving little room for ambiguity or misinterpretation. This precision helps in identifying potential flaws or inconsistencies early in the design phase.
- **Correctness**: By mathematically proving the properties of the system, formal methods help ensure that the final product meets its intended specifications.
- **Cost Reduction**: Although an additional initial investment, experience shows that formal methods can ultimately lead to cost savings because they allow finding errors in the earlier stages of development, thus reducing the work and the costs of fixing them.
- **Confidence**: Thanks to exhaustiveness, there is greater confidence in having a correct and reliable product.

Still, some weaknesses are present:

- **Decidability Issues**: Different systems require different logics (timed, probabilistic, hybrid) and different goals require different problems (model-checking, synthesis, etc.), and in a lot of cases we end up in a semi-decidable or undecidable problem.
- **State-space explosion problem**: the number of states is exponential on the number of its components (and not only that), thus can be unfeasible to store all of them in memory. To overcome this, many different techniques such as symbolic approaches have been studied.
- **Scalability**: Formal methods may struggle to scale to large, complex systems with numerous components and interactions. As the size and complexity of the system increase, applying these methods may become prohibitive.

- **Expertise**: It requires a user expert on mathematical logic and formal reasoning to define the appropriate abstractions to define the model, the properties, and to implement it.

In summary, formal methods have shown their importance, especially in safety-critical systems, such as those used in aerospace, medical devices, or autonomous vehicles. In the last decades formal methods have gained a lot of interest and their applicability range is enlarging, thanks to better and better algorithms that consent to overcome the above-mentioned obstacles.

2.1 Logics, Automata & Games

In order to fully comprehend what model checking is and how different systems can be modeled, this section will give the definition and some basic notions of the major formalisms used and studied. The section can be divided into two main parts: first will be defined types of temporal logics and then types of automata. Temporal logics are used to write specifications, i.e., behaviours a system should or should not encounter meanwhile automata are used to model systems. Many different logics and automata can be defined, depending on various factors such as the complexity and structure of the system, the computational complexity of logics and automata, the degree of abstraction, etc.

The formalisms here presented either are necessary to express the experimental part (such as the Hybrid Automata) or provide the fundamental basis on which the required complex formalisms are built (like Timed and Stochastic Automata). In particular, our model relies on the following formalisms: a **Hybrid Automaton (HA)** that models the plant, the **Metric Time Interval Logic (MITL)** used to express the properties to be verified, and on **Statistical Model Checking (SMC)** which estimates the probability of satisfying the MITL formulas.

2.1.1 LTL: Linear Temporal Logic

Linear Temporal Logic is a formal language for expressing properties along a discrete linear time sequence. It allows to write *qualitative* specifications of a model. Its syntax extends the classical propositional logic with two temporal operators: next (unary) and until (binary).

Definition 2.1.1 *The **syntax** of (Propositional) Linear Temporal Logic is defined over a set of atomic propositions AP by the following grammar:*

$$\phi ::= p \mid \phi \wedge \phi \mid \phi \vee \phi \mid \neg\phi \mid \mathrm{X}\phi \mid \phi \mathrm{U} \phi \quad \textit{where } p \in AP$$

Definition 2.1.2 *An LTL **structure** $\mathcal{M}$ is defined as a tuple (S, R, L) where S is the set of states, $R \subseteq S \times S$ is the transition relation on states and $L : S \to 2^{AP}$ is the labeling function.*

A infinite sequence of states $x \equiv (s_0, s_1, ...) \equiv (x(0), x(1), ...)$ is called a *run*. An LTL formula ϕ is satisfied if a given infinite run x starts from an initial state s_0 of the structure $\mathcal{M}$ and makes ϕ true. To verify if a run x in $\mathcal{M}$ effectively satisfies ϕ ($\mathcal{M}, x \models \phi$), we introduce the notation of x^i, which is the infinite suffix of the run x starting from position i, formally defined as $\forall i \geq 0 : x^i \equiv (s_i, s_{i+1}, ...)$. Note that $\mathcal{M}, x \models \phi \iff \mathcal{M}, x^0 \models \phi$. The notation of x^i comes in handy to define the semantics of LTL:

Definition 2.1.3 *Given a structure $\mathcal{M}$ and a sequence of states x the* ***semantics*** *of LTL is defined as:*

$$\begin{aligned}
\mathcal{M}, x^i \models p &\iff p \in L(s_i) \\
\mathcal{M}, x^i \models \phi \wedge \psi &\iff (\mathcal{M}, x^i \models \phi) \wedge (\mathcal{M}, x^i \models \psi) \\
\mathcal{M}, x^i \models \neg\phi &\iff \mathcal{M}, x^i \not\models \phi \\
\mathcal{M}, x^i \models \mathrm{X}\phi &\iff \mathcal{M}, x^{i+1} \models \phi \\
\mathcal{M}, x^i \models \phi \mathrm{U} \psi &\iff \exists j \geq i \, (x^j \models \psi \wedge \forall i \leq k < j \, (x^k \models \phi))
\end{aligned}$$

Informally, the semantics of LTL follows from propositional logic, plus the one of next and until:

- **next**: if Xp holds at time t implies that p will be valid in the next step $t+1$.

- **until**: if pUq holds at time t implies that p will hold in all time steps from t -at least- until $t+j$, where q will become true, for some $j \geq 0$. Note that the until can be defined as strong (U$^{\exists}$) or weak, where the strong version forces the existence of such j, thus q must eventually become true. Again, the until can be defined as either strict (U$_{>}$) or non-strict (U$_{\geq}$), where the strict version forces $j > 0$, hence, to validate pUq, q should become true in a future time step. The until here described is assumed w.l.o.g. to be strong and non-strict: $U^{\exists}_{\geq}$.

Moreover, if a formula ϕ holds for any possible run x, then ϕ is said to be valid in $\mathcal{M}$ and is short-handed with $\mathcal{M} \models \phi$. Similarly, if there exists a run x that makes ϕ false, then ϕ is not valid a formula for $\mathcal{M}$: $\mathcal{M} \not\models \phi$. Note that, being not valid doesn't mean being unsatisfiable, it does just imply that there *exists* a run x that falsifies ϕ.

In our experiments, we did not explicitly use LTL. However, we introduced it because MITL logic is a timed extension of LTL. Explaining how MITL works becomes easier if the reader is already familiar with LTL. In Uppaal, the symbol for the next operator is `X`, while for the until operator is `U`. Moreover, it is also possible to use the modal operators, immediately below described, of possibility $\Diamond$ and necessity $\Box$, with the symbols `<>` and `[]`.

Expressing Modal Logic in LTL

The well-known modal operators of possibility $\Diamond\phi$ and necessity $\Box\phi$ can be derived by combining the LTL operators, thus they don't add any expressive power. In temporal logic, the possibility is also known as "sometimes in the future" or "finally" (Fϕ) and necessity as "always in the future" or "globally" (Gϕ). Table 2.1 contains a summary of these concepts.

$$\begin{aligned}
\textit{sometimes}\,(\textit{possibility}) &: \Diamond\phi \equiv \mathrm{F}\phi \equiv (\textit{true}\,\mathrm{U}\phi) \\
\textit{always}\,(\textit{necessity}) &: \Box\phi \equiv \mathrm{G}\phi \equiv \neg(\mathrm{F}\neg\phi) \equiv \neg(\textit{true}\,\mathrm{U}\neg\phi)
\end{aligned}$$

operator name	temporal symbol	modal symbol
next	$\mathrm{X}\phi$	$\bigcirc\phi$
until	$\phi\mathrm{U}\psi$	
sometimes	$\mathrm{F}\phi$	$\Diamond\phi$
always	$\mathrm{G}\phi$	$\Box\phi$

Table 2.1: Temporal operators of Linear Temporal Logic with their modal counterparts.

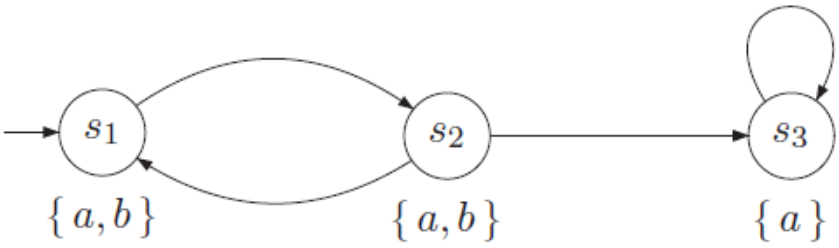


Figure 2.1: An example of a Kripke Structure with propositional letters $\{a, b\}$ (Baier, Katoen 2008) [8]

Example

Consider the structure in fig. 2.1 taken from [8] with two propositional letters $\{a, b\} = AP$, three states $\{s_1, s_2, s_3\} = S$, and s_1 as the unique initial state. Below each state is depicted the set of propositions valid in it: in s_1 and s_2 both a, b hold while in s_3 only a. Note that state s_3 is a so-called *sink* state, i.e., has no exiting edges towards other states, thus if it is reached it cannot be left.

Trivially, $\Box a$ is a valid formula because a holds in any state: $\mathcal{M} \models \Box a$. Differently, $\Box b$ is not a valid formula ($\mathcal{M} \not\models \Box b$) because there exists at least one run (actually infinite runs) that can reach s_3 and falsify $\Box b$, such as $s_1 s_2 s_3^\omega$. On the contrary, the run $(s_1 s_2)^\omega$ makes $\Box b$ true: $\mathcal{M}, (s_1 s_2)^\omega \models \Box b$. Thus, the formula $\Box b$ is satisfiable but not valid in $\mathcal{M}$. An example of an unsatisfiable formula is $\Box\neg b$ because b holds in the unique initial state s_1. The formula $b\,\mathrm{U}\,(a \wedge \neg b)$ states that b will always hold until a point in which exclusively a will hold. The run $(s_1 s_2)^\omega$ is a *counter-example* for not validating the latter formula in $\mathcal{M}$.

2.1.2 CTL: Computational Tree Logic

Computational Tree Logic, also known as Branching Time Temporal Logic, has a semantic similar to the one of LTL. The key difference is that CTL considers multiple discrete sequences of time together. In fact, each state can have multiple successors, and so, starting from a unique starting state s_0 the time in CTL should be represented as a tree and not as a sequence as in LTL. To generate such a tree, CTL enriches the syntax of LTL with two additional operators, known as *path quantifiers*: "for All paths" ($\mathrm{A}\phi$) and "there Exists a path" ($\mathrm{E}\phi$). The temporal operators of LTL (next and until) are instead called *state quantifiers*. Basically, the $\mathrm{A}\phi$ operator forces ϕ to be true in all possible different future paths, meanwhile $\mathrm{E}\phi$ affirms that in at least one future ϕ holds. Note that, CTL is not a generalization of LTL. That's because CTL forces a strict alternation (every state quantifier must be preceded by a path quantifier). Thus, the two logics are used to express properties of different models.

Definition 2.1.4 *The **syntax** of (Propositional) Computational Tree Logic is defined over a set of*

atomic propositions AP by the following grammar:

$$\phi ::= p \mid \phi \wedge \phi \mid \phi \vee \phi \mid \neg\phi \mid \mathrm{AX}\phi \mid \mathrm{A}[\phi \mathrm{U} \phi] \mid \mathrm{EX}\phi \mid \mathrm{E}[\phi \mathrm{U} \phi] \;\; \textit{where } p \in AP$$

Definition 2.1.5 *A CTL **structure** $\mathcal{M}$ is defined as a tuple (S, R, L) where S is the set of states, $R \subseteq S \times S$ is the transition relation on states and $L : S \to 2^{AP}$ is the labeling function.*

The semantics of CTL is defined depending on the type of state: for path-formulas is $\mathcal{M}, x \models \phi$ where x is an infinite sequence of states (as defined for LTL), and is exactly identical to the LTL's one (see 2.1.3), meanwhile for state-formulas is $\mathcal{M}, s_0 \models \phi$ where $s_0 \in S$ is the initial state and is below given.

Definition 2.1.6 *Given a structure $\mathcal{M}$ and an initial state $s_0 \in S$ the **semantics** of CTL state formulas is defined as:*

$$\begin{aligned}
\mathcal{M}, s_0 \models p &\iff p \in L(s_0) \\
\mathcal{M}, s_0 \models \phi \wedge \psi &\iff (\mathcal{M}, s_0 \models \phi) \wedge (\mathcal{M}, s_0 \models \psi) \\
\mathcal{M}, s_0 \models \neg\phi &\iff \mathcal{M}, s_0 \not\models \phi \\
\mathcal{M}, s_0 \models \mathrm{E}\phi &\iff \exists x = (s_0, ...)\, (\mathcal{M}, x \models \phi) \\
\mathcal{M}, s_0 \models \mathrm{A}\phi &\iff \forall x = (s_0, ...)\, (\mathcal{M}, x \models \phi)
\end{aligned}$$

CTL*

CTL* is a generalization of both CTL and LTL. Intuitively, CTL* has the same operators and semantics that CTL has, but a larger syntax: it removes the alternation rule on path and state quantifiers.

In general, CTL has been the most used because it offers a good trade-off between expressiveness and efficiency. In fact, it has been proved that Model Checking of LTL and CTL* is $PSPACE$-complete while CTL is P-complete and thus -theoretically- much more efficient. However, LTL Model Checking is practical as well because its complexity is $2^{O(|\phi|)} * O(|\mathcal{M}|)$, which means that it is linear on the size of the structure -typically large- and exponential in the size of the formula -typically small-.

Example

Consider again the model presented in figure 2.1. A trivially valid formula is $\mathrm{A}\Box a$, which means that in any possible future a will always hold. The similar formula is $\mathrm{A}\Box b$ is instead unsatisfiable because s_3 is reachable from s_0 and in s_3 b doesn't hold. A satisfiable formula is $\mathrm{E}\Box b$, which holds only for the run $(s_1 s_2)^\omega$. A more complex (valid) formula states that $\mathrm{A}\Diamond(\neg b \to \mathrm{A}\Box\neg b)$, i.e., in every possible future if eventually b becomes false then in all futures b will always remain false. The latter is trivially true because $\neg b$ holds only in s_3, and being s_3 a reachable sink state, $\neg b$ will hold forever in s_3.

2.1.3 DTMC: Discrete Timed Markov Chains

A Markov Chain (MC) describes a system whose states change over time according to a stochastic process. The fundamental property of a MC is its *memoryless* nature: the next state depends only on

the current state (the path that led to the current state is irrelevant). Thus, a MC is the obvious choice for encoding a stochastic system. A Discrete Timed Markov Chain (DTMC) is a MC enriched with time, modeled as a discrete set.

Definition 2.1.7 *Let AP be a finite set of atomic propositions. A DTMC is a tuple* $\mathcal{M} = (Q, q_0, R, L)$ *where* Q *is a finite set of states,* $q_0 \in Q$ *is the initial state,* $R : Q \times Q \to [0, 1]$ *is the transition probability function where* $\forall q \in Q \left[\sum_{q' \in Q} R(q, q') = 1\right]$ *and* $L : S \to 2^{AP}$ *is the labeling function that identifies the set of the true atomic propositions in each state.*

In a DTMC, time is implicitly encoded: it increases by one after taking a transition. A DTMC can be represented using a graph as in example 2.2, later shown in section 2.1.5. Yet, the common way to represent the transition function of a DTMC is by using a transition matrix, which is a variation of a classical adjacency matrix of a graph where the cell $\mathcal{M}[i, j] \in [0, 1]$ encodes the probability of taking the transition $i \to j$. Therefore, row i represents the probability of leaving node i to some node j and sums to 1. On the other hand, column j shows from which set of nodes the node j is directly reachable: $\{i : \mathcal{M}[i, j] > 0\} \subseteq Q$. However, the sum of a column is not a meaningful value.

Although our implemented model is not stochastic, we introduced the DTMC because Statistical Model Checking (SMC) was originally developed to verify properties of stochastic models like DTMCs. SMC, however, is fundamental for our model. We present SMC in section 2.4, and introducing it using examples with DTMCs is straightforward. Therefore, we included a brief introduction to DTMCs here.

2.1.4 MDP: Markov Decision Processes

Markov Chains assume the environment to be entirely stochastic. MCs are not applicable anymore in systems where both stochastic and non-deterministic transitions are present. In the latter case, Markov Decision Processes (MDP) should be used. MDPs model discrete-time stochastic control processes and are suitable for decision-making situations.

Definition 2.1.8 *Let AP be a finite set of atomic propositions. A MDP is a tuple* $\mathcal{M} = (\Sigma, Q, q_0, R, L)$ *where* Σ *is a finite alphabet (the set of actions)* Q *is a finite set of states,* $q_0 \in Q$ *is the initial state,* $R : Q \times \Sigma \times Q \to [0, 1]$ *is the transition probability function where* $\forall q \in Q \, \forall a \in \Sigma \left[\sum_{q' \in Q} R(q, a, q') \in \{0, 1\}\right]$ *and* $L : S \to 2^{AP}$ *is the labeling function that identifies the set of the true atomic propositions in each state.*

Note that, the transition matrix is 3-dimensional, and the sum over a starting state can also be zero (a state may not have stochastic transitions).

As mentioned in section 2.1.7, our model does not have probabilistic transitions and is therefore not a stochastic model. We introduced MDPs for a similar reason as DTMCs: both are simple formalisms that encompass a few key features, making them easy to explain. Specifically, MDPs are likely the most widespread and simple models that incorporate the presence of a controller, and synthesizing a controller is the ultimate goal of our model. In section 2.5.1, we describe various types of strategies that can be developed, using MDPs to provide straightforward examples that highlight these differences.

2.1.5 PCTL: Probabilistic Computational Tree Logic

Probabilistic Computational Tree Logic has been introduced to deal with probabilistic systems. Stochasticity consents to verify *quantitative* properties: instead of yielding a binary answer to the satisfiability of a given formula, the aim is to *estimate* its probability. In order to do so, the syntax and the semantics of CTL are enriched with a probability operator: $\mathrm{P}_{\sim\theta}(\psi)$. The semantics of $\mathrm{P}_{\sim\theta}(\psi)$ is the following: $\mathrm{P}_{\sim\theta}(\psi)$ is true in state s if the overall probability that ψ holds in any path starting from s is $\sim\theta$. The rest of the PCTL syntax and semantics recall the classical CTL logic.

Definition 2.1.9 *The **syntax** of Probabilistic Computational Tree Logic is defined over a set of atomic propositions AP by the following grammar:*

$$state\ formula\ \phi ::= p \mid \phi \wedge \phi \mid \neg\phi \mid \mathrm{P}_{\sim\theta}(\psi)$$

$$path\ formula\ \psi ::= \mathrm{X}\phi \mid \phi \mathrm{U}^{\leq t}\phi \mid \phi \mathrm{U}\phi$$

where $p \in AP$, $t \in \mathbb{N}$ is a time bound, $\theta \in [0,1]$, $\sim \in \{<, \leq, =, >, \geq\}$.

Computing the probability P of a finite path $s_0 s_1 ... s_n$ is straightforward: $P(s_0 s_1 ... s_n) = \prod_{i=1}^{n} \mathbb{P}(s_{i-1} s_i)$. Then, the probability of satisfying a formula ψ from state s_0 is the sum of all the paths starting from s_0 that satisfy ψ: $\mathrm{P}_{\sim\theta}(\psi) = \sum_{\mathcal{M}, s_0 s_1 ... s_n \models \psi} P(s_0 s_1 ... s_n) \sim \theta$, where n depends on the path operator of ψ: $n = 1$ for $\mathrm{X}\phi$, $n = t$ for $\phi \mathrm{U}^{\leq t}\phi$, and $n = \infty$ for $\phi \mathrm{U}\phi$.

For instance, consider the example in figure 2.2: the DTMC $\mathcal{M}$ represents a coin-flipping process. Initially, there is a 50% chance of getting either tails or heads. However, as soon as the coin lands on tails, it is replaced with an unfair coin with tails on both sides (thus a 100% probability of landing on tails). Let ϕ be "the probability of getting tails in almost 3 steps is greater than 60%", i.e., $\phi = \mathrm{P}_{>0.6}\mathrm{F}^{\leq 3}(tails)$. Let ψ be the path formula in ϕ: $\psi = \mathrm{F}^{\leq 3}(tails)$. $\mathrm{P}(\psi)$ is given by the probability of the two paths $s_0 s_2$ and $s_0 s_1 s_0 s_2$. $\mathrm{P}(\psi) = P(s_0 s_2) + P(s_0 s_1 s_0 s_2) = 0.5 + 0.5 * 1 * 0.5 = 75\%$. Thus, $\phi = \mathrm{P}(\psi) > 60\% = 75\% > 60\% = true$.

Note that, in the syntax of PCTL the path quantifiers $\mathrm{E}\psi$, $\mathrm{A}\psi$ are not present. The probability operator itself acts in place of them: $\mathrm{P}_{>0}(\psi)$ is the identical probabilistic counterpart of $\mathrm{E}\psi$ and similarly $\mathrm{P}_{\geq 1}(\psi)$ for $\mathrm{A}\psi$, but in a weaker way. This weakness is due to the fact that if a formula is satisfiable with a probability that tends to zero as the number of steps tends to infinity, the formula is considered false.

To clarify, consider again the example in figure 2.2. If we ignore the probabilities, the model $\mathcal{M}'$ becomes non-deterministic instead of stochastic. Consider the CTL formula $\mathrm{AF}(tails)$ which states that for all possible paths soon or later the coin will land on tails. Asking a CTL model checker whether $\mathcal{M}', s_0 \models \mathrm{AF}(tails)$ would return false, with counter-example $(s_0 s_1)^\omega$. Now, let's reintroduce again the probabilities and consider the original model $\mathcal{M}$. The PCTL counterpart of the CTL formula $\mathrm{AF}(tails)$ is $\mathrm{P}_{\geq 1}\mathrm{F}(tails)$. Unlike the previous case, we know that there is a 50% chance of getting tails on the first flip. If it fails, we go back to state s_0, and again the probability of getting tails is 50%. The probability of never getting tails is 50% for the first flip times 50% for the second, and so on, which is $(0.5)^n$, where n is the number of flips. Clearly, the limit of such a probability tends to zero as n approaches infinity. Thus, $\mathcal{M}, s_0 \models \mathrm{P}_{\geq 1}\mathrm{F}(tails)$ is true, and no counterexample would be found.

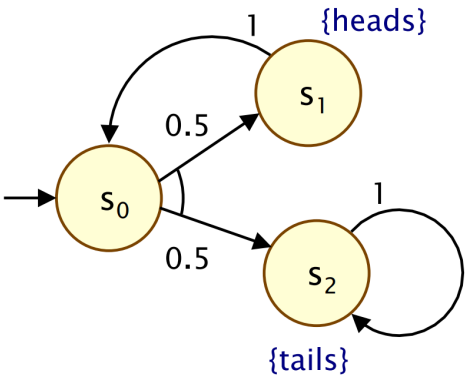


Figure 2.2: The model represents a coin-flipping process. The initial state is s_0, where we hold a fair $(50\% - 50\%)$ coin. We flip the coin, and if the outcome is heads, we restart the process in s_0. If the outcome is tails, we switch to an unfair coin with tails on both sides. We continue flipping the unfair coin, which will always land on tails. (Michaelmas 2011)

Note that, in PCTL has been introduced the bounded until $p\mathrm{U}^{\leq t}q$, which has (almost) the same semantics of the unbounded version but forces q to eventually become true within time t, where t is the number of steps (transitions). In classical CTL the bounded version of the until can be easily derived but is not so necessary because the unbounded version -much more powerful- can be verified as well. Differently, as stated in 2.4.1, solving the unbounded version of the until in the probabilistic setting is still an issue.

2.1.6 TA: Timed Automata

ω-regular languages are widely used in model checking and formal verification of reactive systems. They provide a formal specification language for expressing system requirements and properties, which can be automatically verified. They can capture qualitative features such as liveness and fairness, essential for reasoning about infinite executions. By abstracting from time, ω-regular languages do not allow checking quantitative features. Timed automata have been introduced by Alur and Dill in 1994 [4], and represent an extension of classical finite state automata, necessary for modeling the behavior of real-time systems.

The idea is to annotate Büchi (or Muller) automata with temporal constraints, using a finite set of real-valued clocks X. Time increases inside states for all clocks at the same rate but can vary in different states. Transitions take place instantly and reset to zero its associated subset of clocks. To enforce a timed behaviour on a system, clock constraints (guards and invariants) can be defined combining Boolean operator with comparisons of the form $x \sim c$ where x is a clock value, $\sim \in \{<, \leq, =, >, \geq\}, c \in \mathbb{Q}$ is a rational constant. A transition can then be executed only if its guard holds at time t, i.e., the interpretation of the clocks at time t makes the guard true. Moreover, invariants force the system to leave the state s as soon as the invariant $I(s)$ becomes false.

While a Büchi automaton recognises a ω-regular language (a set of ω-regular words), a timed automaton recognises a timed language. A timed language is defined by a set of timed words. A timed word is a couple of the form (σ, τ) where $\sigma \in \Sigma^\omega$ is an ω-regular word, τ is an infinite timed sequence. A timed sequence $\tau = \tau_1\tau_2...$ is an infinite sequence of time values $\tau_i \in \mathbb{R}$ with $i > 0$, that satisfy (i) **monotonicity**: τ increases strictly monotonically, and (ii) **progress**: τ diverges to infinity.

Definition 2.1.10 *Given a set of clocks X, $\Phi(X)$ is a set of clock constraints φ defined by the following grammar:*

$$\varphi := x \leq c \mid c \leq x \mid \neg\varphi \mid \varphi \wedge \varphi$$

where $x \in X$ is a clock and $c \in \mathbb{Q}$ is a rational constant.

Definition 2.1.11 *A timed automaton $\mathcal{A}$ is a tuple $\langle \Sigma, L, L_0, X, I, E \rangle$ where Σ is a finite alphabet (the set of labels), L is a finite set of locations, $L_0 \subseteq L$ is a set of initial locations, X is a finite set of clocks, $I : L \to \Phi(X)$ is a set of clock constraints (invariants), $E \subseteq L \times \Sigma \times L \times 2^X \times 2^{\Phi(X)}$ is the transition relation such that $\forall(s, a, s', \lambda, \varphi) \in E$ the edge has form $s \to^a s', \lambda \in 2^X$ is the set of clocks to be reset, $\varphi \in 2^{\Phi(X)}$ is the clock constraint that specifies if the edge is enabled (guard).*

Definition 2.1.12 *Given a timed automaton $\mathcal{A}$, its **semantics** is defined via a Transition System $S_{\mathcal{A}}$:*

- ***states** have form (s, v) with $s \in L$ and v is a clock interpretation $v : X^{|X|} \to \mathbb{R}^{|X|}$ s.t. $v \models I(s)$*
- ***initiation condition**: (s_0, v) is an initial state **if** $s_0 \in L_0$ and $v = \mathbf{0}$*
- ***consecution condition**, which has two types:*
  - *delay transition: $(s, v) \to^\delta (s, v(X) + \delta)$ **if** $\forall\delta' \in [0, \delta]\,((v(X) + \delta') \models I(s))$*
  - *action transition: $(s, v) \to^a (s', v[\lambda := 0])$ **if** $v \models \varphi$ and $(s, a, s', \lambda, \varphi) \in E$*

where $v \models \varphi$ stands for "v satisfies φ"

Note that TA have an uncountably infinite state-space: although definition 2.1.11 allows for a compact representation, in reality, as definition 2.1.12 suggests, any pair $(s \in L, v \in \mathbb{R}^{|X|})$ represents a different state. To overcome this issue can be defined a equivalence relation on interpretations called region equivalence, which allows the grouping of different interpretations that share the same behaviours. Thanks to the finiteness of the relation, can then be constructed a finite timed automaton called the region automaton that has an exponential size in the total length of the clock constraints in $\mathcal{A}$[4].

In the above definition have been omitted the acceptance conditions. To obtain a Timed Büchi (or Muller) Automaton (TBA or TMA) one needs only to define the set of final states according to their classical semantics. The well-known and limiting result is the undecidability of the universality problem (checking if automaton $\mathcal{A}$ is equivalent to $\mathcal{A}_{univ}$, which recognizes any possible timed language) [4]. From this result follows the non-closure under complementation of the timed languages. Thus, as explained in 2.3, only safety properties can be verified for a general timed automaton, unless the liveness clause is definable via a deterministic TMA (DTMA). A DTMA recognizes a strict subset of the sets of timed languages, but is fully decidable: the complementation of a DTMA is straightforward. So, it is sufficient that the liveness property $\mathcal{A}_\phi$ is encoded in a DTMA. Even if the system $\mathcal{A}_S$ is modeled with a non-complementable automaton (DBA/DMA), checking whether $L(\mathcal{A}_S) \subseteq L(\mathcal{A}_\phi)$ is decidable.

We discussed Timed Automata for two main reasons: (i) are fundamental for expressing simple real-time processes, and (ii) can be easily generalized to Hybrid Automata. Given that our experimental model is based on a Hybrid Automaton, introducing Timed Automata first facilitates the subsequent explanation of Hybrid Automata and highlights their complexity.

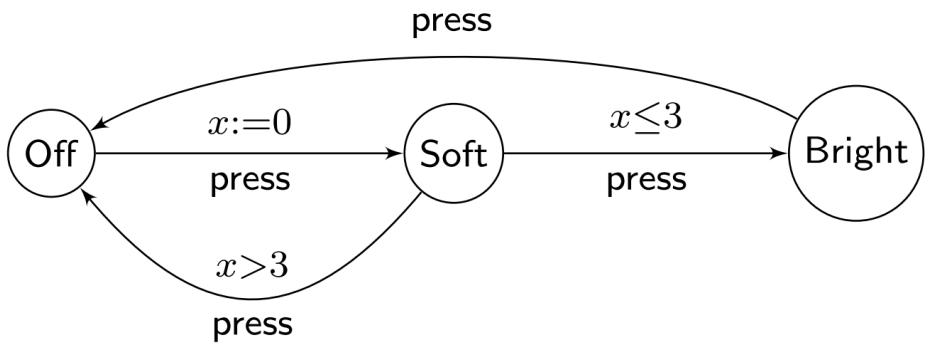


Figure 2.3: An example of a timed automaton that models an intelligent light controller.

Example

Consider the timed automaton in picture 2.3. It models an intelligent light controller: by clicking just once will turn on the light. By clicking twice in a span of 3 time units it will turn on the light (as before) but produce a brighter light. If the second click takes place after 3 time units it will be switched off. The system is modeled with a timed automaton composed of three locations $L = \{Off, Soft, Bright\}$, one action $\Sigma = \{press\}$, one clock variable $X = \{x\}$. With the first click, we reset the clock x and move to location $Soft$. If the second click happens within 3 time instants the enabled transition will be the one with guard $x \leq 3$ towards location $Bright$. Otherwise, when $x > 3$, the light will be turned off by taking the transition towards location Off. Note that, all the locations are invariant-free, i.e., it is allowed to remain indefinitely long inside a location. Two possible traces that show the two possible behaviours of the model are:

$$(Off, 0) \rightarrow^{press} (Soft, 0) \rightarrow^{1.5} (Soft, 1.5) \rightarrow^{press} (Bright, 1.5)$$

$$(Off, 0) \rightarrow^{press} (Soft, 0) \rightarrow^{3.5} (Soft, 3.5) \rightarrow^{press} (Off, 3.5)$$

2.1.7 MITL: Metric Interval Temporal Logic

Metric Temporal Logic (MTL) is the timed extension of LTL with clock variables, designed to specify properties of real-time systems. MTL is designed to verify properties of timed systems, where in each location time increases in a continuous manner. So, even though time is still considered as a linear time sequence (as in LTL), now is continuous and not discrete anymore.

Unfortunately, model checking with MTL is undecidable. What makes undecidable MTL is punctuality: MITL is instead decidable because prohibits singular time intervals ($[a, a]$). For example the formula $\phi = \Box(p \rightarrow \Diamond_{=5} q)$ which says that "whenever p holds, q will hold *exactly* 5 time instants later" is undecidable (taken from [5]). A decidable logic instead is the Metric Interval Temporal Logic (MITL), the fragment of MTL without punctuation. The formula ϕ cannot be stated in MITL but can be approximated with finite (arbitrary) precision: $\psi = \Box(p \rightarrow \Diamond_{(4.9,5.1)} q)$.

Definition 2.1.13 *The **syntax** of Metric Interval Temporal Logic is defined over a set of atomic propo-*

sitions AP by the following grammar:

$$\phi ::= p \mid \phi \wedge \phi \mid \phi \vee \phi \mid \neg\phi \mid \phi \mathrm{U}_I \phi$$

where $p \in AP$ and I is an interval in one of the following forms $[a,b], [a,b), (a,b], (a,b)$ with $a, b \in \mathbb{N} \cup \{\infty\}$ s.t. $a < b$. By allowing $a = b$ we obtain the MTL syntax (which is undecidable). Note that the modal operators $\Diamond$ and $\Box$ are definable in the usual manner, as explained for the untimed case. On the contrary, the next operator is not included in the syntax of MITL since it is designed for discrete-time models. The semantics for the continuous-time until is reported while for the other operators follow from the classic Boolean rules.

Definition 2.1.14 *The **semantics** of the Metric Interval Temporal Logic is defined over a timed word $\rho = (\sigma, \tau)$, and a time instant i as:*

$$\rho, i \models \phi \mathrm{U}_I \psi \iff \exists j \in (i, |\rho|) \; s.t. \; \rho, j \models \psi; \; \tau_j - \tau_i \in I; \; \forall k \in (i,j) \, \rho, k \models \phi$$

In practice, the semantics is the same as the one for LTL, but forces ψ to hold for the first time inside interval I. Moreover, the classic untimed operators can be easily defined by setting $I = [0, \infty)$ (for the strict version) or $I = (0, \infty)$ for the non-strict. Model checking a MITL formula against a timed automaton is EXPSPACE-complete [5]. If the MITL formula contains only disequalities rather than intervals ($\mathrm{U}_{<a}$ and $\mathrm{U}_{>a}$, or more precisely, $\mathrm{U}_{[0,a)}$ and $\mathrm{U}_{(a,\infty)}$) then the model checking complexity drops to PSPACE-complete.

We presented MITL because it plays a key role in our experimental part. In fact, we expressed the properties in MITL, and the model using a hybrid automaton. Note that, both are not stochastic. But, as later shown, to verify properties on a Hybrid Automaton we used Statistical Model Checking (SMC). Statistical Model Checking estimated the probability of a formula given a set of finite simulations. In Uppaal, the "statistical queries" are essentially MITL queries but preceded by a probabilistic operator `Pr`. Thus, those queries are not properly MITL queries, but they share the syntax and semantics of classical MITL.

Moreover, previously we introduced PCTL to present how probabilistic logics work. As just said, our queries are not PCTL but MITL (with a stochastic interpretation due to SMC). Yet, the latter two, even though are different, are bonded in the spirit: estimating the probability of a property.

2.1.8 HA: Hybrid Automata

Hybrid Automata (HA) have been introduced to deal with the most complex systems imaginable: hybrid systems (HSs). As one could expect, for HA even the emptiness problem is undecidable, thus SMC is the only possible way to estimate properties on general hybrid automata. HSs represent the most abstract and general formalism for a system: discrete and continuous variables can be defined and can interfere with each other. Discrete variables are fixed inside a state and are updated in a transition while continuous variables (the clocks) have the opposite behaviour: evolve inside states and are fixed -if not reset- in transitions. Moreover, each clock can have its own rate that can depend on both the discrete and the continuous variables, amounting to ordinary differential equations (ODEs). The evolutions of

the continuous variables inside states are referred to as the *trajectories*.

In 1996, Henzinger published *The theory of Hybrid Automata* [31], he was the first to introduce and study Hybrid Automata. Each location of the automaton is labeled with predicates *inv*, and *flow*: the first defines the invariant, and the second defines how each variable evolves inside the location. The automaton contains a set of continuous variables $X = \{x_1, ..., x_n\}$. During the continuous change inside a location, the first derivate of the variables is represented as $\dot{X} = \{\dot{x}_1, ..., \dot{x}_n\}$, while the value of the variables at the end (when the location is left) is written as $X' = \{x'_1, ..., x'_n\}$.

The following definition is instead taken from David et al [22], in which they define a *delay* function F that has the same role as the *flow* predicate. Each state is labeled with $F(v, \delta) = v'$ which takes in input the current interpretation v and a period δ and computes the value of each clock after a time-span δ. In Uppaal, both F and I do not have a restricted syntax: for example, any user-defined function, any clock value, or any ODE can be potentially used. From a theoretical point of view, F and I are normally assumed to be first-order formulae. Apart from the invariant I and the delay function F, the remaining syntax and semantics of Hybrid Automata recall the ones of Timed Automata.

Definition 2.1.15 *A Hybrid Automaton $\mathcal{M}$ is a tuple $\langle \Sigma, L, l_0, X, I, E, F \rangle$, where Σ is a finite alphabet (labels, actions) L is a finite set of locations, $l_0 \in L$ is an initial location, X is a finite set of continuous variables, $I : L \to \Phi(X)$ is the set of clock constraints per state (invariants), $E \subseteq L \times \Sigma \times L \times 2^X \times 2^{\Phi(X)}$ is the transition relation (switches) such that $\forall (s, a, s', \lambda, \varphi) \in E$ the edge has form $s \to^a s', \lambda \in 2^X$ is the set of clocks to be reset, $\varphi \in 2^{\Phi(X)}$ is the clock constraint that specifies if the edge is enabled (guard), $F : L \times \mathbb{R}^{|X|} \times \mathbb{R}_{\geq 0} \to \mathbb{R}^{|X|}$ is the delay function of each location.*

Notation Note: if the trajectory of a continuous variable corresponds to a straight line, such continuous variable is also referred to as a:

- **memorycell** if $\forall\, l \in L\ (x' = x)$ or, equivalently, $\dot{x} = 0$.
- **clock** if $\forall\, l \in L\ (x' = x + t)$ or, equivalently, $\dot{x} = 1$.
- **stopwatch** if $\forall\, l \in L\, \exists\, a_l \in \{0, 1\}\ (x' = x + a_l * t)$ or, equivalently, $\dot{x}_l \in \{0, 1\}$.
- **two-slope** if $\exists\, a, b \in \mathbb{N}_{>0}\, \forall\, l \in L\ (x' = x + a * t) \vee (x' = x + b * t)$ or, equivalently, $\dot{x}_l \in \{a, b \mid a, b > 0\}$.
- **skewed clock** if $\forall\, l \in L\, \exists\, a_l \in \mathbb{Z}_{\neq 0,1}\ (x' = x + a_l * t)$ or, equivalently, $\dot{x}_l = a_l$, $a_l \neq 0, 1$.
- **linear** if $\forall\, l \in L\, \exists\, a_l \in \mathbb{Z}\ (x' = x + a_l * t)$ or, equivalently, $\dot{x}_l = a_l$.

where a is a constant over all states, while a_l is a constant depending on the location l (and similarly for $\dot{x}$, and $\dot{x}_l$).

A binary memorycell (can be set only to 0 or 1 in edges) is called a **proposition**. Note that all the above types are particular cases of the last one, so are all linear. Moreover, memorycells represent discrete variables, and, as clocks, are also stopwatches.

Since, as already mentioned, model checking is undecidable for Hybrid Automata, many subclasses have been studied and developed with (semi-)decidable results. The subclasses of a Hybrid Automaton can be partitioned into two main categories: linear or non-linear hybrid automata.

Prior to detailing the subclasses of HA, we specify that the remaining part of this section is unrelated to the experimental part. The below-listed subclasses may offer the possibility to formally verify the properties of a system, but are not as powerful as general hybrid automata. Our model, as later presented, seems to belong to the class of general HA, or, anyway, would be a very hard (if not impossible) task to transform it into a subclass of HA having a (semi-)decidable reachability problem. The reason is that we deal with many continuous variables, and the ODE of a variable y may depend both on the value of other continuous variables $x_1, ..., x_n$ or on the derivative of such variables $\dot{x}_1, ..., \dot{x}_n$. Moreover, in this system of ODEs we have as well cases of circular dependencies. Thus, looking for rewriting rules that may simplify the set of ODEs seems unfeasible. Instead, with Statistical Model Checking we can estimate properties with no additional effort on modifying the model.

Linear Hybrid Automata (LHA)

Linear Hybrid Automata restrict HAs by forcing the invariant and the jump predicates to be defined using linear constraints on the variables of the form $a_0 + a_1x_1 + ... + a_nx_n \leq b$, or, using the vectorial notation: $a^T x \leq b$. Note that, a conjunction of finitely many linear constraints defines a polyhedron. Moreover, each flow predicate must be of the form: $\forall x \in X \, \forall l \in L \, \exists a_l \in \mathbb{Z} \; (x' = x + a_l * t)$. Despite these constraints, the reachability problem on LHAs is undecidable [3]. By increasing the restrictions, from a LHA one can obtain:

- **Rectangular HA (RHA)**: RHAs force invariant, flow, and jump predicates to be rectangles, which are cartesian products of intervals rather than arbitrary linear expressions. Each flow condition must satisfy a rectangular bound, i.e., each rate of change must have a lower and an upper bound: $\exists a, b \in \mathbb{R} \, \forall l \in L \; (a_l \leq \dot{x} \leq b_l \wedge a_l < b_l)$. Unfortunately, the reachability problem is already undecidable for a RHA with one two-slope clock [33]. There exists an even stricter class of RHAs, known as **Initialized Rectangular HA** (IRHA) for which the reachability problem is PSPACE-complete, as shown in [33]. IRHAs are RHAs with two additional constraints: (i) values of two variables with different flows are never compared; (ii) when the flow of a variable changes, the value of that variable must be reinitialized (reset).

- **Multi-Rate HA (MRHA)**: A Multi-Rate HA (MRHA) is a linear hybrid system where its variables can be either skewed clocks or propositions. An n-rated HA is a multi-rate whose skewed clocks can proceed at maximum n different rates. Unfortunately, is sufficient to have a 2-rate timed system to make undecidable the reachability problem [3]. A LHA is *simple* if in location invariants and transition guards the comparisons of the variables consist only of inequalities against integers ($x \leq k \vee k \leq x, k \in \mathbb{Z}$). By combining simplicity with multi-rate we obtain **Simple Multi-Rate HA** (SMRHA) for which the reachability is decidable [3].

Note that, **Timed Automata** (see section 2.1.6) can be seen as a particular case of both a Simple Multi-Rate and of a Rectangular Automata where variables can be only clocks or propositions, and for which the reachability problem is PSPACE-complete [4].

Non-Linear HA (NLHA)

Differently from the linear HAs which allow only simple trajectories, NLHAs allow more complex non-linear flows. A well-studied class of non-linear HAs are **O-minimal HA** (O-MHA). An infinite model $\mathcal{M} = (M, <, ...)$, containing a total order relationship $<$, is an *o-minimal structure* if and only if any subset $X \subseteq M$ is definable via a finite union of intervals and points. A theory $\mathcal{T}$ is o-minimal if contains only o-minimal models. An o-minimal theory $\mathcal{T}$ is decidable if admits the quantifiers' elimination. An o-minimal hybrid automaton (O-MHA) must have invariant, flow, and jump predicates defined using an o-minimal theory. For every O-MHA there exists a finite bisimulation, and, if the theory used to define the predicates is decidable, then also the bisimulation is decidable. The reachability problem can be reduced to a finite bisimulation, and, if the above assumptions hold, it is decidable, as proved by Lafferriere et al [38].

2.2 Main problems

Many different types of problems can be defined in mathematical logic. Depending on the expressive power of the logic on which we desire to solve them, their complexity will change as well. In fact, the more expressive the logic is the more computationally complex the problem will be, up to undecidability. In order to define the various problems, we first need to characterize two fundamental concepts:

Finite Transition System (FTS) is a finite state graph where any possible configuration of our model is represented as a state, and the transitions are used to model events that alter the state. A classical FTS is the **Kripke Structure**, a graph enriched in each state with a set of valid propositional formulas in it, which correspond to the properties that hold.

Temporal Logic provides the syntax for stating formulas, and determining which properties can be checked. The most classical examples are Linear Temporal Logic (LTL) and Computational Tree Logic (CTL), which widen the syntax of propositional logic with temporal operators.

Finally, here they are:

Satisfiability (SAT) given a formula ϕ formed by a set of boolean variables X, verify if there exists an assignment $\alpha : X \to \{0, 1\}$ that satisfies ϕ:

$$\exists \alpha \, (\phi(\alpha) = True)$$

Validity (VAL) given a formula ϕ, verify if for any assignment α the formula $\phi(\alpha)$ is valid (a *tautology*):

$$\forall \alpha \, (\phi(\alpha) = True)$$

Logical Equivalence (LE) given two formulas ϕ_1, ϕ_2, verify if for any α their outcome is equal:

$$\forall \alpha \, (\phi_1(\alpha) \leftrightarrow \phi_2(\alpha))$$

Model Checking (MC) given a structure S and a formula ϕ, verify if S models ϕ:

$$S \models \phi$$

Synthesis given a formula ϕ, find and construct S that models ϕ:

$$\texttt{return}\, S : S \models \phi$$

Satisfiability has been the first well-studied problem in the area and still has a lot of applications. SAT is known to be an NP-hard[1] problem. There are many approaches to solve the problem such as constraint programming or logic programming. Note that since

$$VAL(\phi) \Leftrightarrow \neg SAT(\neg\phi) \tag{2.1}$$

the validity problem is the complementary problem of SAT and has a coNP-hard complexity. Logical Equivalence seems a harder problem w.r.t. to satisfiability and validity but actually, LE can be reduced to VAL:

$$LE(\phi_1, \phi_2) \Leftrightarrow VAL(\phi_1 \leftrightarrow \phi_2) \Leftrightarrow \neg SAT(\neg(\phi_1 \leftrightarrow \phi_2)). \tag{2.2}$$

So, basically, by solving the SAT problem we get for free an algorithm for both VAL and LE. For this reason, the scientific community has mainly focused on solving the SAT problem.

Model checking is easier than the three precedent problems, as it requires the input structure to be built already. Model checking can be seen as the circuit value problem in propositional logic, which is P-complete. Moreover, in first-order logic (FO), being simpler allows for model checking to be decidable while SAT, VAL, and LE are all undecidable. Nevertheless, model-checking algorithms rely on identity 2.1: if the unsatisfiability of a *bad* property ϕ during any possible execution of the model S is established, then ϕ will never happen, i.e., the model S is *safe* against ϕ.

Differently, synthesis is the most complex problem of the listed above since it aims to automatically construct the program that satisfies the set of the given properties. The idea is to mathematically define all the properties our program should satisfy and the synthesizer should produce the code that solves the problem. A synthesizer can be used also in optimization tasks: by defining a game in which are given a set of possible actions and a goal function to minimize, the synthesizer should find the winning strategy that always employs the best action. Given the hard-theoretically goal, in a lot of formal languages, the problem is undecidable.

2.3 Model Checking

Model checking (MC) has proven to be one among the most popular solutions in verification theory. The idea is to check whether a program satisfies the set of properties or if any of them could eventually become false. Before the development and application of such a method, traditional methods like peer review and testing were -and still are- widely used to verify software. However, they did not provide

[1] Assuming ϕ is a propositional formula.

satisfactory results in critical systems. *Peer review* is based on a static analysis of the code made by a team of software engineers. As for [8] peer review is used in almost 80% of projects and allows to identify between 31% and 93% of the defects. Still, a lot of bugs and problems are almost impossible to catch in this way. *Testing* is a semi-automatic strategy to verify dynamically how the program behaves: a set of inputs with its expected output is given and is verified against the obtained output. Testing is the most natural and intuitive way to check software: after a piece of code is written, it is executed to verify if its behaviour matches our expectations. The problem is that is very expensive (typically 30% to 50% of the whole project budget) and it does not even guarantee actual correctness! Correctness is assured only for the small subset of inputs of the test set, but exhaustive testing is unfeasible. A model checker instead, is a tool that explores all possible system states in order to verify a set of properties. Any possible system behaviour is described with a model in a mathematically precise and unambiguous manner. In particular, MC can be decomposed into four main phases, as shown in picture 2.4:

1. Definition of the **system**'s formal specification. Often, already at this stage, a lot of ambiguities and inconsistencies are discovered due to previous informal specifications.

2. Definition of the specification's **properties**: which bad properties must never occur (*safety* requirements) and which good properties should eventually occur. (*liveness*)

3. **Design and implementation** of the above on a tool that enables model checking.

4. **Execution** of the system to verify the properties. If any fails, a *counterexample* will be returned specifying which sequence of actions led to that unexpected state. With it, the user can replicate the trace error and accordingly adapt the model or the property (go back to step 1. or 2.).

As informally introduced above, two fundamental types of properties are checked on a system:

- **Safety condition**: in any possible infinite sequence of state (*trace*) σ in system S, property ϕ must *never* happen. Essentially, bad situations must never occur. Safety is a universal condition: ϕ must be false in *all* possible states.

- **Liveness condition**: in any infinite trace σ in system S, property ϕ must *eventually* happen. Liveness properties are used to assure that good situations will occur sooner or later. Liveness is an existential condition: ϕ will hold in *some* states in the future.

The classical formalism to verify such properties is a Büchi or Muller Automaton. In the years many extensions have been developed such as timed, stochastic, and hybrid automata. Before showing how to verify these properties, some notation is introduced: capital letters $A, B, ...$ for automata, greek letters $\phi, \varphi, ...$ for formulas and properties, $L(A)$ for the language recognized by automaton A and $\overline{L(A)}$ for the complement of that language. Formulas are used to express the properties of a system. A formula ϕ can also be turned into its corresponding automaton A_ϕ, the one that recognizes the language $L(A_\phi)$, given by all paths that satisfy the formula ϕ. So, A_S, A_ϕ represent the automaton of the system and the property. The procedure to check if a safety property ϕ holds consists of verifying whether $L(A_S) \cap L(A_\phi) \neq \emptyset$. Differently, to check a liveness property it is necessary to show that $L(A_S) \subseteq L(A_\phi) \Leftrightarrow L(A_S) \cap \overline{L(A_\phi)} = \emptyset$. For this reason, from an algorithmic point of view liveness conditions are

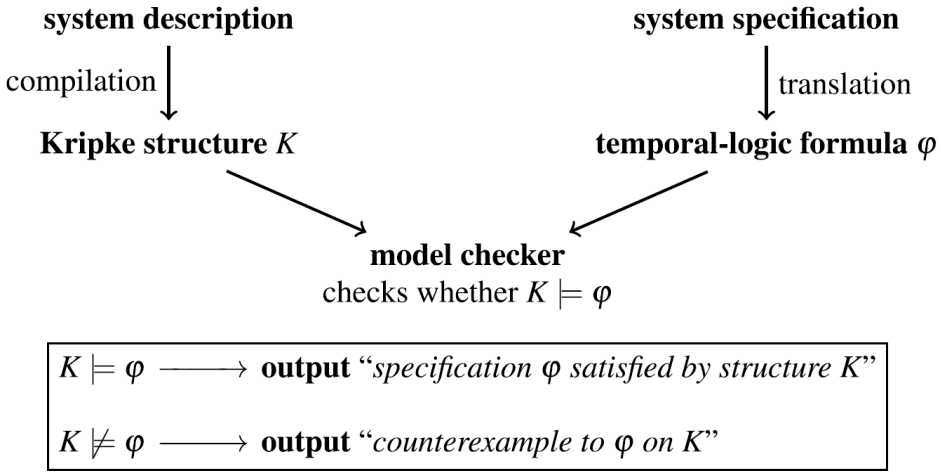


Figure 2.4: The model checking workflow (Clarke 2018) [20]

		reachability		language inclusion	
system	property	decidability	complexity	decidability	complexity
Muller / Buchi Automata	LTL	YES	PSPACE	YES	PSPACE
Muller / Buchi Automata	CTL	YES	P	YES	P
Muller / Buchi Automata	CTL*	YES	PSPACE	YES	PSPACE
Timed Automata	TA	YES	PSPACE	NO	
Timed Automata	DTMA	YES	PSPACE	YES	PSPACE
Timed Automata	MITL	YES	EXPSPACE	YES	EXPSPACE
Discrete Timed Markov Chain	PCTL	YES	PSPACE	YES	PSPACE
Hybrid Automata	(any)	NO		NO	

Figure 2.5: Summary of Model Checking complexity

at least as complex as safety properties. Typically, all formalisms in this area are closed under union and intersection making the verification of a safety condition a decidable problem. Unfortunately, not all formalisms are closed under complementation yielding -in these cases- an undecidable problem. An example of the latter is Timed Automata, which in their most general definition are not closed under complementation.

2.3.1 Computational Complexity

Before concluding the section on Model Checking, a brief summary of its complexity is given. Model Checking is a decidable problem in many (but not all) formalisms. It is P-complete for CTL and PSPACE-complete in most logics. Remember that reachability is used to solve the emptiness problem and thus verify safety conditions, while language inclusion is necessary to guarantee liveness conditions. The results shown in picture 2.5 are taken from [44] for LTL, CTL, CTL*, [5] for MITL, [4] for TA, [52] for the DTMC, [31] for HA.

2.4 Statistical Model Checking

Simulation-based approaches -widely used in engineering methods-, have gone unused and ignored for reasoning about formal specifications for decades. The reason is pretty simple: a finite set of simulations

cannot guarantee the truth of a formula for any infinite execution but only for those finite simulations. However, as systems have become more complex, traditional formal verification techniques have shown their limits in handling large-scale systems, and sometimes they are not fully or even partially applicable. Moreover, classical model checking only allows the verification exclusively of *qualitative* properties. In decision-making tasks, it is often necessary to evaluate properties *quantitatively*, such as understanding the probability of an event to occur rather than just acknowledge that there is a remote possibility that it could happen.

Statistical Model Checking (SMC) is inspired by the Monte-Carlo method and emerged as a response to these challenges. By definition, SMC represents a compromise between testing and classical formal method techniques. Classical MC suffers from the well-known *state space explosion problem* due to the need to apply the cartesian product of the various components of the system [8]. SMC instead, being simulation-based, doesn't require performing such products, making SMC far less memory and time-intensive. Another advantage is that SMC can be applied to any observable system, including *black-box* ones -having an unknown structure- or with infinite-state systems. In fact, the statistical analysis needs exclusively a sampling procedure, which can be either applied via simulations or executions. Once the sampling is performed, the desired properties can be verified -with a certain degree of confidence- using *hypothesis testing*. Still, some cons can be highlighted: (i) the verification is probabilistic, (ii) reducing the uncertainty degree leads to a fast growth of the sample size, (iii) it works only for purely probabilistic systems (non-determinism is not allowed). Before delving into the topic, note that most of the content in this section is taken from the notable survey work on SMC by Agha and Palmskog [1].

2.4.1 Verification techniques

To statistically verify the properties of a Markovian system, modeled via a DTMC, formulas are expressed with a stochastic temporal logic (PCTL). Initially, to check quantitative specifications were applied numeric techniques and, later on, numeric-symbolic manipulations. Both techniques allow to estimate with good precision if a PCTL-formula ϕ holds for a given DTMC $\mathcal{M}$ in the initial state s_0 [52]. Such methods require an explicit representation of the finite-state model and have scalability issues.

However, a SMC problem is *well-defined* if the system is observable, it does not require knowing its structure. By looking at *paths* -infinite sequence of states- and *traces* -finite prefixes of paths- certain conclusions can be drawn. Ideally, SMC is not so different from MC: given a structure S and a formula ϕ both aim to verify if ϕ holds in S. In practice though, the underlying algorithms are completely different: MC tries to solve the reachability problem meanwhile SMC combines simulations with *hypothesis testing*.

Hypothesis Testing

In inferential statistics, there are two major paradigms: the frequentist and the Bayesian approach. Here the focus will regard the frequentist approach. Let $p = \mathrm{P}_{\geq\theta}(\psi)$, H_0 be the *null hypothesis* and H_1 be the alternative hypothesis on the value of p. In hypothesis testing H_0 is assumed to be true until the observed data leads to its rejection.

$$H_0 : p \geq \theta \qquad H_1 : p < \theta$$

	Decision	
Truth	*Accept* H_0	*Reject* H_0
$p \geq \theta : H_0\ true$	correct ($> 1 - \alpha$)	type I error ($\leq \alpha$)
$p < \theta : H_0\ false$	type II error ($\leq \beta$)	correct ($> 1 - \beta$)

Figure 2.6: The contingency table for Binary Hypothesis Testing

As in the confusion matrix shown in figure 2.6, four possible outcomes are possible depending on the truth value of p against the prediction of the value of p. If truth and prediction match the answer is correct. Otherwise, a type I error occurs if we reject H_0 when it is true (a false negative). H_0 is rejected if the estimated probability of H_0 to be true is $\leq \alpha$. The probability $1 - \alpha$ is called the **confidence level**. Lastly, a type II error happens if we accept H_0 when it is false (a false positive). Similarly, H_0 is not rejected if the estimated probability of H_0 to be false is $\leq \beta$. The probability $1 - \beta$ is called the **power of test**.

Moreover, a third parameter δ can come into play. When $p \approx \theta$, the probability of incurring either a type I or II error increases. To mitigate such a risk, an *indifference region* of size δ can be added around the value of θ. The indifference region indicates that, in the face of uncertainty, neither H_0 nor H_1 are rejected or accepted.

$$H_0 : p \geq \theta + \delta \qquad H_1 : p \leq \theta - \delta$$

Sampling

Being based on simulations, generating a diverse and statistically significant set of samples is a crucial matter. Here hypothesis testing is used to determine if there is enough evidence to support the hypothesis. There are two main sampling strategies:

- **Fixed-Size Sample Testing**: as the name says, the size n is predetermined. A way to estimate the necessary n to achieve the required precision is the so-called *Simple Sampling Plan* (SSP). The main drawback of a SSP is that the existence of an analytical formula to obtain the optimal size $m \leq n$ is unknown in general. Therefore, with SSP the optimal m is overestimated with n, limiting the efficiency.

- **Sequential Testing**: follows an incremental approach. A small-fixed batch b of samples is drawn and then the test is computed *on-the-fly*. The procedure is then repeated until, at step i, a size of $i * b = n \geq m$ is reached and the test is passed. This simple strategy reduces the number $n - m$ of useless samples computed with respect to a SSP. Still, its applicability is limited by the availability of obtaining the samples *on-demand*. With sequential testing, exists an approach to compute the optimal m, based on the calculation of cumulative likelihood ratios. The latter is called *Sequential Probability Ratio Test (SPRT)* but the optimality result holds only under certain assumptions.

Identifying Rare Properties

When dealing with stochastic simulations, a common question is "How to find unexpected behaviours that happen rarely due to an improbable sequence of actions?". Of course, analyzing any possible

infinite trace with a statistical model checker is impossible. To increase the likelihood of analyzing any trace, the following techniques are employed to favour unlikely scenarios:

- **Curtailed SSP** takes two parameters: the maximum number of failures k, the sample size n. Simply, after the evaluation of the samples, the system is considered safe if and only if the number of failures detected in the sample is $\leq k$.
- **Importance Sampling** is a variance reduction technique that allows to focus the sampling process on the region of interest. So, instead of sampling following the probability of the original stochastic system, a biased distribution is used to increase the chances of seeing improbable events. Then, to ensure an unbiased sampling estimator, the outcomes are weighted using the likelihood ratios. The effectiveness of this method allows SMC to be efficient and scalable: lots of rare and unexpected behaviours can be identified without using a huge sample size.
- **Importance Splitting** is an alternative to importance sampling. The idea, in this case, is to exploit *conditional probabilities* by decomposing a single simulation trace into a *series* of paths, where each path corresponds to different conditional events, allowing a more accurate estimation of rare properties.

Checking Complex Properties

With respect to classical CTL, the PCTL extends its syntax to the probability operator and the bounded until operator. Given the set of samples checking the bounded until is straightforward. Checking $\mathrm{P}_{\geq\theta}(\psi)$ is easy as well. There are two cases that require more care: the nested probabilistic operator and the unbounded until operator.

- **Nested Probabilities** $\mathrm{P}_{\geq\theta_1}(\mathrm{P}_{\geq\theta_2}(\psi)\ \mathrm{U}^{\leq \mathrm{t}}\ \phi)$. Solving the innermost probabilistic operator $\mathrm{P}_{\geq\theta_2}(\psi)$ (P_2 for brevity) is easy, the outermost one (P_1) instead represents a hypothesis test on P_2 and not directly on the system. So, since a hypothesis test on samples is verified with some error by definition, checking an additional property on top of another hypothesis causes an increase in the error margin. To obtain a reliable outcome, two possible methods can be used, and both have limited applicability. In fact, they are suitable if the stochastic system can be represented as a Markov Chain, which isn't always the case.
  - **nesting the tests** approach is pretty intuitive: to achieve the desired error bounds α_1, β_1 for the outermost formula P_1, then the innermost bounds α_2, β_2 for P_2 should be stricter: $\alpha_2 < \alpha_1 \wedge \beta_2 < \beta_1$ [53]. The downside of this idea is that the sample size grows exponentially with respect to the number of tests.
  - **combining numeric-symbolic and SMC** seems a good compromise: the innermost formula is checked with numeric-symbolic methods, expensive but precise, and the outermost with the classical SMC approach [52]. This combination has several reasons: (i) numeric-symbolic methods are complete, thus path formulas are correctly estimated without error, (ii) using SMC for the outermost formula gives more scalability. However, this strategy does not apply to black-box systems.

- **Unbounded Until** $\psi = \phi_1 \, \mathrm{U} \, \phi_2$. An algorithm to check the unbounded until in the classical CTL settings is well-known and is part of any CTL model checker. It can be verified because infinite traces are analyzed exhaustively and completely. The thing is, this approach cannot be easily re-engineered in PCTL because SMC relies on samples, which of course are finite. Many attempts have been made but the problem is still considered open.

2.5 Controller Synthesis & Games

Model checking, as detailed in section 2.3, has shown its effectiveness in many different domains. Given a system with its programmed controller, model checking allows checking if the controller avoids the environment leading to an undesirable situation. But, rather than verifying if a proposed controller is correct, why not just define the desired property and let a program to automatically find out how that property can be satisfied? Such a problem is called synthesis but it can be further distinguished in program synthesis or controller synthesis.

Since the emergence of AI in the 1950s **program synthesis** has always been considered the holy grail of Computer Science. Ideally, the programmer only tells the computer *what* the software should do, rather than specifying *how* to do it, freeing himself from the implementation details. The goal of program synthesis is not just to verify if a program meets certain requirements, as in model checking, but to construct the program itself! However, program synthesis is undecidable in general, and can result in a harder task than writing the program itself. Program synthesis is not part of this thesis, but the interested reader can consult [30] for more details.

The goal of **controller synthesis** does not regard writing code, but finding the optimal strategy for the controller that handles the system, in order to satisfy (or maximize) the objective function. The strategy should specify how to limit the internal behavior forcing the system to behave in a certain way, regardless of the actions of the environment. Controller synthesis is strictly related to *game theory*: finding a strategy can be seen as a game where *we* (the controller) play against the *environment* in a 2-player game. If there is no *winning strategy* for the controller, the output should be a negative answer. Otherwise, when such a strategy exists, it is returned as output. In which form it should be returned depends on the type of the strategy, of the system, of the controller, etc.

Controller synthesis can also be considered a sub-branch of **control theory** from a more mathematical and theoretical point of view rather than an engineering perspective. Control theory is a field of control engineering and applied mathematics that focuses on the analysis and design of controllers that govern the behavior of dynamical systems in engineered processes and machines. Control theory combines optimization and solving techniques with domain-specific knowledge, and relies also on human intuition, heuristic methods, manual tuning, and analytical resolutions.

Games are used to model and analyze situations where multiple agents (referred to as *players*) interact with each other under a set of rules to achieve specific objectives. In planning and decision-making problems many different types of games have been studied. Games can be purely deterministic (with n players), or also present stochasticity (referred to as the $1/2$ player). In particular, Chatterjee et al. [19] propose the following classification of games, depending on their parameters:

- **number of players**: $1/2$: Markov Chains, 1: non-deterministic state-transition systems, $1\,1/2$:

Markov Decision Processes, 2: game graphs, $2\frac{1}{2}$: stochastic games.

- **players' knowledge**: simple (*turn-based*) games (one action at a time) or *concurrent* games (players move simultaneously and independently)
- **winning objectives**:
  - **qualitative** (ω-regular): finite (reachability or safety) or infinite (liveness, Büchi or parity conditions)
  - **quantitative**: discounted, total or limiting average reward
- **winning criteria**:
  - **qualitative**: sure winning, almost-sure winning (with probability 1), limit-sure winning (with probability $1 - \epsilon$ for any arbitrary small ϵ)
  - **quantitative**: exact probability of winning, expected reward

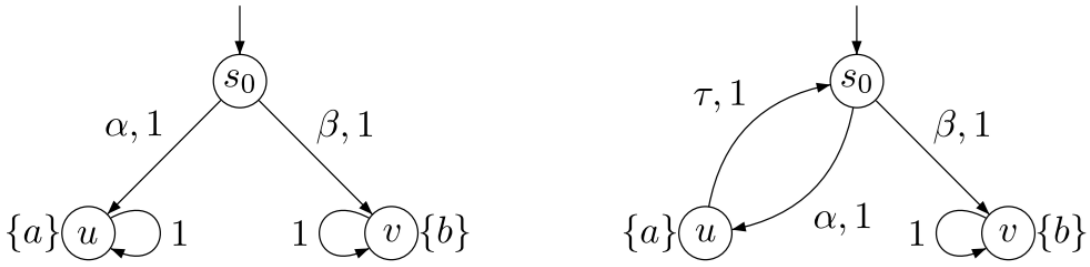
Moreover, additional distinctions regard the degree of observability of the environment (fully, partial, none) and being either zero-sum (agents have opposite utility functions) or general-sum (agents have independent utilities). The problem of synthesizing a controller can be reduced to finding a strategy that wins for the controller against the environment, in a 2-players game. Informally, for a given game, a strategy is a rule that tells the controller which action to perform when a variety of possible actions are enabled in a given state. A strategy is said to be *winning*, if, no matter what the opponent does, there is always a counter-action that leads to victory.

2.5.1 Types of strategies

As for Baier et al. [7], there are two different binary categorizations of strategies, that lead to four different approaches:

- **deterministic (D) or randomized (R)**: in the deterministic case there is a unique arc labeled with a for each state s, while in the random case, the next action will depend upon a probability distribution.
- **markovian (M) or historical (H)**: in the first case the next state is based only on the current state (*memory-less*), otherwise also the precedently visited states can influence.

The four strategy types MD, MR, HD, and HR form a hierarchy. Deterministic strategies can be reduced to random strategies where for each couple (s, a) there is a unique arrival state s' with probability 1. Similarly, a markovian strategy can be seen as an historical one, where the history is limited to the current state itself. Note that, if a given strategy is a solution, then it is a solution also for the other types of strategies that generalize it. This means that an MD-strategy -also denoted as *simple*- is a solution for any type, and that MR- and HD-strategies are also HR-strategy. In general, simpler strategies are preferred, especially markovians over historicals, because there is no need of additional memory to store the past states. Nevertheless, depending on the formula may be necessary to use randomized, historical, or both approaches to solve some specific problems. For example, consider the MDP (a) in figure 2.7,



(a) randomization helps **(b)** history helps

Figure 2.7: Two MDPs with propositional letters $AP = \{a, b\}$ and action-set $\Sigma = \{\alpha, \beta, \tau\}$ (Baier et al., 2004) [7].

and the PCTL specification $\phi = \mathrm{P}_{\geq 0.5}(\Diamond a) \wedge \mathrm{P}_{\geq 0.5}(\Diamond b)$. Clearly, $\Diamond a$ or $\Diamond b$ can be satisfied but not both in the same run. By making use of stochasticity, the strategy in s_0 should randomly (with probability 50-50%) choose one of the two actions, thus satisfying ϕ. On the contrary, see example (b) of the same figure, and the PCTL specification $\psi = \mathrm{P}_{\geq 1}(\Diamond a) \wedge \mathrm{P}_{\geq 1}(\Diamond b)$. Here, a historical strategy is necessary to satisfy ψ: the first time s_0 is visited should be performed action α and the second time action β.

2.5.2 Timed Games

To detail how games can be applied in controller synthesis, from now the focus will regard timed automata, which are more suitable to represent real-time systems. Timed games generalize untimed games including in the analysis timing constraints, in the exact same way that timed automata generalize ω-regular automata. In timed games, a strategy for the controller should not just choose *which* action to perform but also *when* doing it. In particular, assuming a 2-player game arena (the controller and the environment), both can either execute an action or just *wait*. So, synthesizing a controller for a timed automaton is indeed a harder process (w.r.t. to the untimed): besides predicting the consequences *after* his action at time t, the controller should also consider the case that the environment decides to take a transition *before* him at time $t' < t$. Given the above classification of games, untimed games are typically turn-based while timed games are intrinsically concurrent. The following definitions are based on the work of Cassez et al. in 2005 [17].

Definition 2.5.1 *Given a Timed Automaton (TA) $\mathcal{A} = \langle \Sigma, L, l_0, X, I, E \rangle$ (TA def. 2.1.11), a Timed Game Automaton (TGA) $\mathcal{G} = \langle \Sigma_c \cup \Sigma_u \cup \{\lambda\}, L, l_0, X, I, E \rangle$ is a TA where the set of labels (the actions) $\Sigma = \Sigma_c \cup \Sigma_u$ is partitioned into controllable Σ_c (under our control) and uncontrollable Σ_u (under the control of the environment).*

Note that action λ represents the wait action (not taking any transition) and is the only one at the disposal of both the controller and the environment. Ahead of defining a strategy, we'll refer to $\texttt{Runs}(\mathcal{G})$ as the set of all runs in $\mathcal{G}$ reachable from the initial state $(l_0, v[X] = \mathbf{0})$, and to $\texttt{last}(r)$ as the last state of a finite run r.

Definition 2.5.2 *Given a TGA $\mathcal{G} = \langle \Sigma, L, l_0, X, I, E \rangle$, a strategy f over $\mathcal{G}$ is a partial function from $\texttt{Runs}(\mathcal{G})$ to $\Sigma_c \cup \{\lambda\}$, s.t. for any finite run r if $f(r) \in \Sigma_c$ then $\texttt{last}(r) = (l, v) \rightarrow^{f(p)} (l', v')$.*

Observe that (i) in timed games there is no stochasticity and thus timed strategies are all deterministic (D) by definition, (ii) f is a function from finite runs -and not from states- which makes it historical (H). So, the given definition of f is a strategy of type HD. To force a strategy to be markovian (M) one just needs to change the definition of f taking as input states rather than runs or, equivalently, to check whether $\forall r, r' \, [\texttt{last}(r) = \texttt{last}(r') \Rightarrow f(r) = f(r')]$.

Since for TA checking liveness properties is undecidable is general [4], the strategies studied have qualitative and finite winning objectives which correspond to the reachability and safety problems.

Definition 2.5.3 *Given a TGA $\mathcal{G}$ and a set of states $K \subseteq L \times \mathbb{R}^X$, the reachability (resp. safety) control problem consists of finding a strategy f s.t. f enforces to reach (resp. not reaching) K in $\mathcal{G}$ or otherwise affirm its non-existence.*

In 1998, Asarin et al. [6], were the first to propose an algorithm to solve the problem, based on backward fix-point computations of the set of winning states. Although correct, this solution lacks efficiency and scalability. In 1999, Henzinger and Kopke [32] proved that the reachability control problem on timed automata is EXPTIME-hard (for both discrete-time and dense-time controller, see section 2.5.4). Later on, in 2005, Cassez et al. proposed a different approach, an on-the-fly algorithm that explores the state-space forward via a symbolic state-space known as the simulation graph [17]. A year later, Behrmann et al. implemented the latter algorithm and included it into Uppaal as an extension called Uppaal-TiGa (Timed Games, see section 7.1.2) [11].

2.5.3 Rectangular Hybrid Games

Hybrid Games are conceptually similar to Timed Games but in the more general setting of Hybrid Automata. In 1999, Henzinger and Kopke [32] proved that the *sampling-control* reachability problem is EXPTIME-complete also for rectangular automata with nondecreasing or bounded variables (and not just for Timed Automata). This result seems to contradict the undecidability of the reachability problem for rectangular hybrid automata (RHA) (recall section 2.1.8). The key is the "*sampling-control*" notion, as later detailed in section 2.5.4. Briefly, the *sampling* restriction forces the controller to operate in a discrete-time setting (while if we admit a dense-time controller the synthesis -as verification- is undecidable). In particular, while inside the states the continuous variables are free to evolve in a dense-time model ($t \in \mathbb{R}^+$), the controller can force transitions between locations only at discrete-time instants ($t \in \mathbb{N}$). The idea is to make the controller a *partial observer* of the continuous space (for comparison, think of a person who is blindfolded and every 60 minutes can see the surroundings for a second). Differently from the initialization restriction -which guarantees only a finite language equivalence quotient-, the sampling-control constraint also ensures a finite bisimilarity quotient, and thus, decidability. Note that the decidability of the *sampling-control* reachability problem implies the decidability of the *sampling-verification* reachability problem (in other words... LTL and CTL model checking!). The latter, as shown in [32], is PSPACE-hard. Moreover, the same results obtained for rectangular automata with nondecreasing or bounded variables (verification in PSPACE and synthesis in EXPTIME) apply also for the broader class of *generalized* rectangular automata with nondecreasing or bounded variables.

2.5.4 Discrete-Time vs Dense-Time Control Modes

A real-time model requires a dense representation of time and Timed Automata were designed for this purpose. The reachability problem for Timed Automata is PSPACE-complete. Nevertheless, switching the point of view from the reachability problem (there exists a path from s_0 to s_f) to the more hard control reachability problem (there exists a sequence of actions s.t. I can impose to go from s_0 to s_f) some considerations must be done, as suggested in the work of Cassez et al. [18]. In particular, if the controller should be able to decide to take a transition at any time instance $t \in \mathbb{R}^+$ (dense) or $t \in \mathbb{Q}^+$ (discrete).

In the **discrete-time model**, while the plant is still modeled with dense time, the controller can observe the plant state only at regular time intervals, and only in these moments can impose a transition. The *sampling rate* α (the rate at which the controller checks the plant) can be either *known* (KSR) or *unknown* (USR). In the first case, α is fixed *a priori* while if it is unknown must be discovered, if exists.

In the **dense-time model**, the controller continuously observes the plant and may trigger a discrete transition as soon as a specific state is encountered (during the continuous evolution). In other words, for each location l is given a set of predicates P_l (part of the invariant I_l), and the controller watches such P_l and, if a predicate $p \in P_l$ changes its value, the controller can force a transition. In a location l, the controller can also decide to let time pass without forcing any action; and any amount of time $\delta \in \mathbb{R}^+$ (as long as the invariant holds) can be spent inside location l. The *switch conditions* (the set of predicates $P_l \subseteq I_l$) can be *known* (KSC) or *unknown* (USC). Assuming the controller doesn't trigger any action in l, in the known case the controller cannot influence the time span δ that will pass (since the invariant I_l is fixed), and, for this reason, this type of control is called *time-abstract*. In the unknown case instead, the set of predicates P_l is unknown and, thus, the controller could also add new stricter predicates to the invariant I_l which can force to leave earlier location l. The USC case is also called *polyhedral control* because, to find a solution, the controller may also derive new (polyhedral) predicates that strengthen the invariants.

Note that, KSR, KSC, USC, and USR -in this order- have an increasing degree of complexity, as shown in picture 2.8. Cassez et al. highlight that **dense-time control modes should be avoided** because they could lead to undesired solutions. Consider a controller that finds a solution by imposing the transitions at the following time instances: $0, 1/2, 1, 1\,1/4, 2, 2\,1/8$, etc. Although being non-Zeno (time diverges), a physical realization is clearly impossible. This bad class of solutions can be avoided only by adding a minimum bound ϵ between consecutive control actions thus leading to a discrete-time problem with $\alpha = \epsilon$. Moreover, they also proved that if α is unknown, then the reachability control problem is undecidable even for timed automata.

In our model, later depicted in figure 8.1 of section 8.2, the time control mode is known and discrete. This means that there is a known fixed sampling rate (KSR) which regularly interrupts the continuous evolution, allowing the controller to take over at defined intervals. To do so, amongst the invariants, we added a predicate that checks if the time t is less or equal to the constant `UPDATE_INTERVAL` (and then we reset t when we leave the location).

Actually, we have a kind of KSC to ensure that we leave the `Filtering` location when the amount of feed in the tank reaches the minimum threshold (50 liters). To implement it, the (uncontrollable) transition from the `Filtering` to the `End` location invokes the urgent channel `ASAP` as soon as the guard

	KSR	KSC	USC	USR
Timed automata	$\surd$	$\surd$	$\surd$	$\times$
Initialized rectangular automata	$\surd$	$\surd$	$\times$	$\times$
Rectangular automata	$\surd$	$\times$	$\times$	$\times$

Figure 2.8: Decidability results for safety control problems. KSC stands for "known dense-time switch conditions" while KSR for "known discrete-time sampling rate" and similarly for USC and USR, where U stands for "unknown". Simbol $\surd$ stands for decidable, while $\times$ for undecidable (Cassez et al., 2002) [18].

$V_f \leq$ `VOL_FEED_MIN` becomes true, where the feed volume (V_f) is a continuous variable evolving inside the `Filtering` state while `VOL_FEED_MIN` is a constant. Yet, it is not properly the method described above because the latter predicate is not part of the invariant of the `Filtering` location. Initially, we indeed tried to add the constraint $V_f >$ `VOL_FEED_MIN` in the invariant, but the Uppaal syntax does not allow to specify lower bounds on continuous variables. Then, we were suggested to add the negation of the constraint as a guard on an urgent channel. By citing the Uppaal manual [47]: "*urgent channels are similar to regular channels, except that it is not possible to delay in the source state if it is possible to trigger a synchronisation over an urgent channel*", thus, the transition must be immediately taken. After, it also says that "*Notice that clock guards are not allowed on edges synchronizing over urgent channels*", probably because it is basically a workaround that allows including lower bounds on clocks. Yet, only a warning is given but the simulator works fine with the expected behaviour.

In any case, an alternative approximated solution could be the following. It relies on another discrete time rate interval, defined with the discrete variable `END_INTERVAL` initially set equal to `UPDATE_INTERVAL` (so useless). Then, as we approach the minimum threshold, the variable `END_INTERVAL` is decreased (during the discrete transitions towards the controller), up to a minimum ϵ value. Then, when $t = i *$ `END_INTERVAL` $<$ `UPDATE_INTERVAL` (where i is a counter) we check whether V_f is still greater than `VOL_FEED_MIN`, or we otherwise terminate the process.

3

Chemical and Physical Context

To provide context, this section will start by introducing some basic chemistry definitions. It will also cover various techniques for separating mixtures, with particular attention on filtration methods, which are the focus of our case study. In chemistry, some fundamental definitions are:

Mixture A mixture is a combination of two or more substances that are physically combined but not chemically bonded. The components of a mixture retain their individual properties and can be separated by physical means. Mixtures can be classified into two main categories: homogeneous and heterogeneous. An **homogeneous mixture** is uniform in composition throughout. The different components are not visually distinguishable because they are evenly distributed. An **heterogeneous mixture** has a non-uniform composition, meaning that the different components can be visually distinguished and are not evenly distributed. Moreover, two relevant properties of a mixture are stability and transparency. **Stable** mixtures do not separate under a gravitational force, while unstable ones can distributed evenly by a mechanical force such as shaking, but will eventually settle out. **Transparent** mixtures don't scatter a beam of light passing through, while non-transparent do scatter.

Solution A solution is a homogeneous mixture of two or more substances. In a solution, one substance (the *solute*) is dissolved in another substance (the *solvent*). The particles of the solute in a solution are at the molecular or ionic level, typically with a diameter $< 1\,[nm]$. Solutions are stable (do not settle out) and transparent; and the solutes are not visible. Solutions can't be separated using size-difference methods but -depending on the solution- common separation methods are distillation, evaporation, crystallization, chromatography, and reverse osmosis. An example of a solution is sugar water: the sugar (the solute) is dissolved in water (the solvent).

Dispersion A dispersion is a heterogeneous mixture consisting of multiple phases, typically two. The predominant phase is called *dispersing* (or *continuous*) phase, while the other phases are referred to as *dispersed* phases. Dispersions can be classified based on the size of the dispersed particles into colloids and suspensions. Dispersions are neither stable nor transparent. Separation methods based on the size difference (such as micro- and ultra-filtration) are effective in separating particles, nanoparticles, and droplets dispersed in the solvent. Other techniques are centrifugation, sedimentation, coagulation, and, only for colloids, dialysis.

Colloidal System Colloids are homogenous but cloudy mixtures and lay between suspensions and solutions. Similar to solutions, the particles of colloids do not settle out, however, are not transparent. The dispersed particles are larger than those in solutions, typically between $[1, 1000]\,[nm]$ (up to a micrometer), but not large enough to be visible by an optical microscope. Depending on the phases, colloids can be further distinguished in emulsions, foam, aerosols, and sols. Examples of colloids are fog (aerosol), and mayonnaise (emulsion).

Suspension Suspensions are usually cloudy or opaque, and their particles can be seen with the naked eye or a microscope (are bigger than $1\,[\mu m]$). Suspensions aren't stable, the particles will settle out over time if left undisturbed. A classic example of suspension is oil and water.

3.1 Membrane Separation Processes

Membrane separation processes are a class of separation techniques that rely on the use of semipermeable membranes to separate different components of a dispersion based on their size, shape, charge, or chemical properties; but not solutions, which are homogeneous mixtures. Membranes can be made from various materials, including polymers, ceramics, and metals, and they come in different pore sizes and configurations. The choice of the membrane depends on the size and characteristics of the particles or solutes to be separated (see section 3.3).

The goal of these processes is to separate again the phases of the given dispersion. The initial dispersion is called the *bulk feed*, the filtered dispersion the *permeate*, and the separated solvents the *retentate*. Their unit measure is either the liter $[L]$ or the volume $[m^3]$.

Bulk Feed f (or b) $[L]$ The initial dispersion to be separated by the membranes in two complementary streams: the permeate and the retentate.

Retentate r $[L]$ The portion of the feed dispersion that does not pass through the membrane. It contains the solid components compacted and ready to be disposed of.

Permeate p $[L]$ The feed portion which instead passes through the membrane, cleared of the parts that did not meet the permeability criteria of the membrane.

Depending on the driving force, it's possible to distinguish different classes of approaches [48]:

Pressure-Driven which includes microfiltration, ultrafiltration, nanofiltration, and reverse osmosis. The pressure difference drives the feed solution through the membrane. The size and charge of the particles determine which components pass through the membrane and which are retained.

Concentration-Driven which comprises dialysis and electrodialysis. The gradient concentration difference is established across the membrane so that solutes move from the high-concentration region to the low-concentration one.

Vaporization-Driven which consists of pervaporation and distillation methods. The vapor pressure or volatility difference separates the components.

3.2 Pressure-Driven Processes

Microfiltration, ultrafiltration, nanofiltration, and reverse osmosis are all pressure-driven processes used in various industrial applications, including water treatment. Micro-, ultra-, and nano-filtration use a **size exclusion mechanism** called *straining* to separate the permeate: membranes have tiny pores that allow only small enough particles to pass through them. In reverse osmosis (and partially in nanofiltration) the predominant forces are the differences in solubility or diffusivity rather than straining. Each of these processes differs in terms of the size of particles they can remove and the characteristics of the membranes used.

The filtration capability of a membrane is usually expressed as a **nominal molecular weight cutoff (MWCO) [Da]**. The molecular weight (MW) represents the weight of a single molecule and is calculated via a weighted sum of the atomic mass of each atom appearing in the molecule times the number it appears in the molecule. For example, the molecular weight of the water is $\mathrm{MW}(H_2O) = 2 * w(\mathrm{H}) + 1 * w(\mathrm{O}) \approx 18.0153$ [Da]. The MWCO of a membrane is the lowest molecular weight for which is expected that 90% of the solute is retained by the membrane. In other words, any molecule having a molecular weight $\leq$ MWCO will have a rejection coefficient $\mathbb{R} \geq 90\%$ (def of $\mathbb{R}$ given at equation 6.7). The main characteristics of each process is listed below, ordered from the coarsest to the finest separation method, as summarized in figure 3.1:

Microfiltration (MF) , membranes' pore sizes: $[0.1, 10]$ µm, MWCO: > 100 kDa, operating pressure (TMP): < 2 bar. MF is suitable for separating suspended solids, bacteria, and large colloidal particles. Common applications of MF include water and wastewater treatment, the filtration of beverages (e.g., beer and wine), and the removal of particulates from the air.

Ultrafiltration (UF) , membranes' pore sizes: $[0.01, 0.1]$ µm, MWCO: $[1, 500]$ kDa, operating pressure (TMP): $[2, 10]$ bar. UF is typically used for the removal of proteins, macromolecules, colloids, suspended solids, bacteria, and some viruses from water. It is commonly employed in water and wastewater treatment, as well as in various industrial processes to separate and concentrate proteins, polysaccharides, and other large solutes.

Nanofiltration (NF) , membranes' pore sizes: $[0.001, 0.01]$ µm, MWCO: $[0.2, 10]$ kDa, operating pressure (TMP): $[10, 30]$ bar. NF is particularly effective at removing divalent ions (e.g., calcium and magnesium), most organic molecules, viruses, and multivalent salts. It is used in water softening, desalination, and the removal of color and odor from water. Being able to reject salts, NF presents a relevant osmotic pressure differential; and, for this reason, the surfaces of the membranes have a negative charge. When applicable, NF can be advantageous over RO: it requires a lower pressure, gives a high permeate flux, and has cheaper costs (both initial and operational).

Reverse Osmosis (RO) , membranes' pore sizes: < 0.001 µm, MWCO: < 0.2 kDa, operating pressure (TMP): $[35, 100]$ bar. RO is highly effective at removing dissolved salts, ions, small molecules, and virtually all contaminants from water. It is widely used for desalination, producing high-purity water for various industrial and residential purposes, and concentration of liquids.

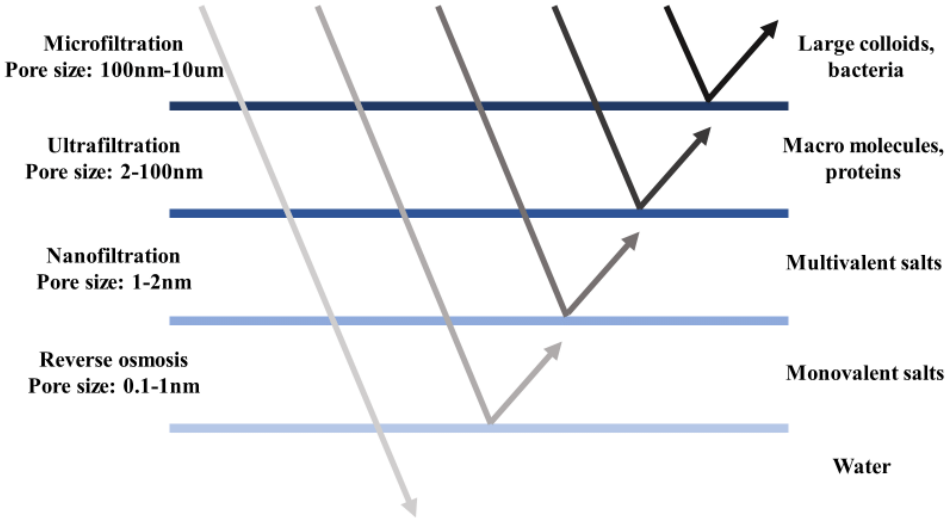


Figure 3.1: Filtration ability of the various pressure-driven processes (Yang et al., 2019) [51].

3.3 Membranes

The current section is based on Sagle and Freeman's introductory article [43] and on Baker's book [9]. Filtration processes can be employed with many different types of membranes. The choice of the membrane depends on the application, for UF the typical pore size is around 0.01 µm. There exist many ways to classify different types of membranes. A common way is by looking at their structure:

- **isotropic**: are uniform in both the composition and the physical nature across the cross-section of the membrane.
- **anisotropic**: are non-uniform over the membrane cross-section, and can be stratified in different layers varying in structure and/or chemical composition. They can be further distinguished in (i) *phase separation membranes*, also called Loeb-Sourirajan -named after their inventors-, which are homogeneous in the composition but not in the structure; and in (ii) *thin film composite membranes*, which are heterogeneous in both.

Another classification possibility is based on their porosity:

- **porous**: membranes that do have pores. MF, UF, and NF membranes are indeed porous. These membranes can be divided again according to their structure: (i) *microporous*: have a rejection strategy exclusively size-based but are susceptible to fouling of species with sizes comparable to that of the pores, (ii) *asymmetric*: have a dense layer (*skin*) on one side and a more porous structure on the other side. Asymmetric membranes get fouled more rarely and are designed to filter species with similar dimensions, exploiting the selectivity and permeability properties of the membrane.
- **non-porous**: membranes that do not have pores, also called dense or impermeable. Note that, RO membranes, even though they have -very tiny- pores, are considered non-porous.

Moreover, membranes can be categorized using their material type:

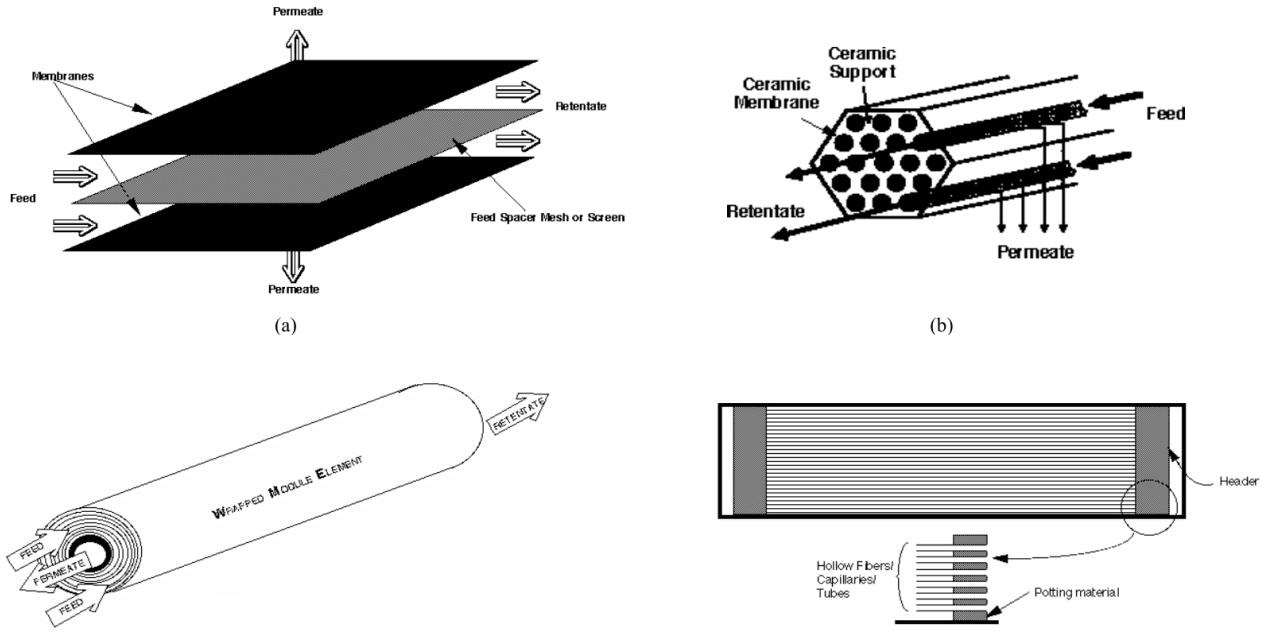


Figure 3.2: Membrane modules of type (a) plate and frame, (b) tubular, (c) spiral wound, (d) hollow fiber modules (Sagle et al, 2004) [43].

- **synthetic organic polymers**: there exist a plethora of possible polymers. Organic membranes are the most widespread because are cheap and available on a large scale for any possible task. Nevertheless, they have some limitations on the operating conditions such as a restricted range of values for pressure, temperature, or pH. Typically, the polymers used are equal for MF and UF but different from the ones for NF and RO.

- **inorganic**: can be either ceramic or metallic. Ceramic membranes are more resistant to temperature, stress, and chemicals and have longer lifetimes but they are far more expensive and fragile. In alternative, metallic membranes can be employed for specific tasks such as gas separation or high-temperature operations.

Finally, there are four possible module's types (see also picture 3.2):

(a) **plate-and-frame**: is the most simple configuration, and consists of two end plates, a flat sheet membrane, and spacers.

(b) **tubular**: the membrane is located inside a tube, and the feed solution flows through the channels while only the permeate traverse perpendicularly the membrane.

(c) **spiral wound**: the membrane is a flat sheet wrapped around a perforated permeate tube. The feed flows on one side of the membrane and the permeate is collected on the other side and moves towards the center in a spiral fashion. They are used mainly in NF and RO.

(d) **hollow fibers**: can be used for water desalination or waste-water treatment. In the first case, they consist of bundles of hollow fibers in a pressure vessel. In the latter instead, there may not be a pressure vessel and thus the fibers' bundles can be suspended in the feed.

3.4 System Components

The physical components of a general pressure-driven filtration system are the following, while the specifics of the plant used for the experiments are given in section 5.1:

Tank Tanks are used to store the initial feed water.

Membrane Module m or (w, wall) It carries out the filtering thanks to its semi-permeable layers that allow the water to pass through while blocking some substances. Can be used as a singleton or either in series or in parallel.

Feed Pump fp Also known as pressure pump, supplies the feed solution to the membrane system at the desired constant pressure.

Circulation Pump cp It is employed to reintroduce the retentate back into the system, in order to re-filter it again. It is present only in the cross-flow filtration approach and serves the purpose of maintaining at the desired rate the tangential flow.

Pressure Valve it allows to change the pressure of the system.

Control System it allows to monitor and regulate the various parameters, such as pressure, flow rate, and backwashing cycles.

3.5 Main Factors

To allow the reader to understand how filtration works, some preliminary definitions are given, including symbols and unit measures, according to conventions.

(Permeate) Flux J $[L\,/\,m^2\,h]$ The rate at which the fluid flows through the membrane per unit area and hour. Increasing the flux makes the filtration faster, and for this reason, it's **the factor** that is studied in filtration systems.

Total Resistance $R_{tot} = R_m + R_{foul} + R_{cp}$ $[1/\,m]$ Is the resistance to flow applied to a fluid by a porous medium. It's given by the sum of the three factors listed below.

Membrane Resistance $R_m = l/p$ $[1/\,m]$ The membrane resistance is the minimum hydraulic resistance to mass transport present in the best conditions when the membrane is clean and the solute to be filtered does not present any suspended solids. It is defined as l/p where l is the membrane thickness and p is the permeability coefficient. Typically, becomes negligible as time goes on since the other two factors have greater orders of magnitude.

Fouling Resistance R_{foul} $[1/\,m]$ The fouling resistance is the most important one and is due to the fouling phenomena: a part of the retentate instead of following its stream will deposit on the membrane surface obstructing its pores and decreasing the membrane permeability and so the flux. A classical counter-action to fight the flux reduction is to increase the driving pressure incrementally. Four main models try to estimate how fouling evolves through time: pore blocking (complete, partial, or internal) or cake model. Further details on fouling resistance at section 4.4.

Concentration Polarization Resistance R_{cp} $[1/m]$ The concentration polarization is due to a part of the rejected solute by the membrane that builds up at its surface with high concentration. This phenomenon causes a higher concentration on the feed side, resulting in a reduced flux as well as reduced apparent rejection. Differently from fouling, concentration polarization is tipically negligible in ultrafiltration (while relevant in ultrafiltration and reverse osmosis). For details on the concentration polarization see section 4.5.

Volumetric Flow Rate Q $[L\,/\,h]$ It represents the rate at which the volume of the fluid flows per unit of time and area. Can be calculated for each stream: for the feed Q_f, for the retentate Q_r (also known as circulation flow rate), and for the permeate Q_p which corresponds to the flux J multiplied by A, the surface area of the membrane.

Mass Concentration C $[g\,/\,L]$ It is the mass amount of a solute per volume of the solution. It is calculated as the ratio between mass and volume: $C = \frac{m}{V}$. Can be easily obtained for the whole solution: picking a small sample, weighing it, drying it in an oven, and then weighing it again. Instead, is harder to calculate C_i where i is a given species because requires ad-hoc chemistry techniques. As for Q, can be measured for each stream: C_f, C_r, C_p, for the membrane surface C_m (or C_w), and for the gel layer C_g. Note that, while the membrane resistance R_m is a constant, its concentration C_m is not a constant -due to the concentration polarization- but its impact is accounted on R_{cp} and not on R_m.

Pressure P $[kPa]$ In general is the amount of force applied $[N]$ divided by the surface area $[m^2]$. While the Pascal $[Pa]$ is the official unit for pressure in the International System of Units, the Bar $[bar]$ is widely used in everyday life, industry, and even in experiments and published papers. 1 bar is equivalent to 100 kilopascal: $1\ [bar] = 100\ [kPa] = 10,000\ [Pa]$. In this thesis and the source code the unit measure used will be the $[kPa]$. In a filtration system, many pressures come into play:

Inlet Pressure P_{in} $[kPa]$ It is the pressure at which the feed solution is introduced into the filtration system. It can be considered constant since it does not depend on the amount of water present in the tank but on its density, gravity, and height from the ground.

Feed Pressure P_f $[kPa]$ It is the pressure at which the feed solution reaches the membrane (on the upstream side).

Permeate Pressure P_p $[kPa]$ It is the pressure of the permeate as it exits the filtration system (on the downstream side, as for P_r). It can be considered constant since the permeate side is typically open to the atmosphere, resulting in a low-pressure difference w.r.t. to the atmospheric pressure.

Retentate Pressure P_r $[kPa]$ Also known as circulation or outlet pressure, it represents the pressure on the retentate side. Through the pressure valve it can be increased or decreased and its update will be reflected in all the other pressures.

Transmembrane Pressure TMP (or ΔP) $[kPa]$ The TMP is the critical parameter: it represents the driving force that allows the flow of the solvent through the membrane and the rejection of

solutes or particles in pressure-driven processes. For this reason, it is known as the operating pressure, i.e., the pressure difference across the filtration membrane. It can be calculated by averaging the pressure on the feed and the retentate side and then subtracting the permeate pressure. How TMP affects the flux is detailed in section 4.6.

Total Energy E_t $[kWh]$ It is the total energy required by the system, which can be estimated by the energy consumption of the pumps (E_{fp}, E_{cp}). Clearly, the goal is to minimize E_t.

Temperature $temp$ $[°C]$ The temperature is an important factor since it affects the fluid's viscosity and, if not controlled, can damage the plant. In fact, while functioning, pumps generate heat that warms the feed (water) and themselves. It is crucial to monitor the temperature if there is no cooling system, because some components (especially the membranes) may get damaged if they are exposed to extreme temperatures, such as over 50°.

Dynamic Viscosity μ $[Pa\ s]$ It represents the magnitude of internal friction in a fluid, as measured by the force per unit area resisting uniform flow. For a specific element (such as pure water) depends on temperature and pressure, and the values are well-known. In general, in a solution, more factors can come into play. Limiting to the case of wastewater, the third main factor is the concentration, which makes viscosity increase as the concentration increases.

Theoretically, it could be calculated for each of the streams but practically it is needed only for the one on the permeate side μ_p in the mathematical model. For the permeate, it is acceptable to use the well-known values of the viscosity of the pure water, since it is filtered and hence its solid concentration is slim to none. Instead, for the other two streams, this assumption would probably fail.

3.6 Working modes

3.6.1 Dead-End vs Cross-Flow filtration

There are two distinct modes of operation in a filtration system:

- **dead-end filtration** The feed flow is perpendicular to the membrane surface. This setup doesn't provide an escape route for contaminants, leading to their quicker accumulation on the membrane.
- **cross-flow filtration** the feed flow is tangential to the membrane surface. Therefore, the shear flow can help in carrying away parts of the foulants and limiting the cake height, thus maintaining a minimal permeate flux. For this reason, most applications use this mode.

3.6.2 Constant Pressure vs Constant Flux

Understanding -and thus estimating- how the flux will evolve through time is a very complex if not an unsolvable task. Many factors can influence -individually and combined- the flux such as temperature, viscosity, driving pressure, osmotic pressure, concentration, fouling, concentration polarization, membranes' properties, permeability, etc. For this reason, historically ultrafiltration systems have been using

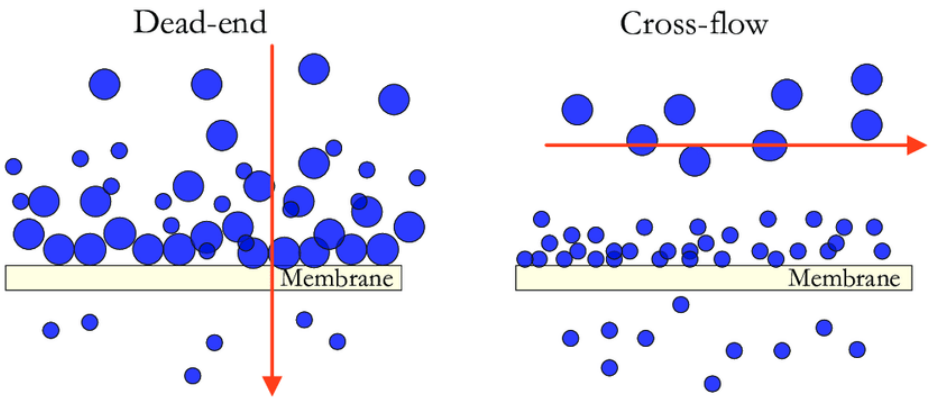


Figure 3.3: Filtration modes: dead-end vs cross-flow (Ruiz-García et al, 2017) [42].

two main operational modes that allow simplifying and estimating the expected behaviour, as shown in picture 3.4.

Until the 1990s most workplaces operated under **constant pressure**. The problem is that pure water flows at high rates without any problem (> 500 $[L/m^2h]$ at 400 $[kPa]$), while a flux of water contaminated with oils or colloids drops quickly. This slowdown is due to fouling and concentration polarization. The time it takes for the flux to stabilize depends on the concentration and ranges from a few minutes to several days. To try to limit the fouling can be employed many techniques: *backflushing*, high-turbulence, *air scrubbing*, asymmetric or *ad-hoc* membranes, and very low flux rates. However, these methods allow to delay and moderate the unavoidable drop in flux rate. To restore the initial flux the unique solution is to stop the system, unload, and clean the membranes. Moreover, note that internal fouling causes irreversible damage to the membrane, thus, even a regular cleaning cycle does not restore the membranes perfectly. As mentioned in section 3.3, inorganic membranes are more expensive but have a longer lifespan while organic ones have opposite pros and cons.

In the mid-1990s, the **constant flux** operation emerged. The idea is to gradually increase the operating pressure as fouling increases in order to compensate for their effects and maintain a constant flux. This approach is opposed to the precedent, where the operating pressure was kept constant and the flux was allowed to slow down. Note that, to put into practice such an idea, is necessary to have some sort of control system that signals the pumps to speed up. Of course, soon or later, the membrane will be fouled and cleaning must be performed. The stopping criteria is an *a priori* fixed upper bound on the TMP value: when it reaches the chosen value the membrane is considered fouled and the system is stopped. Typically, this operation modality uses very low values for TMP and flux.

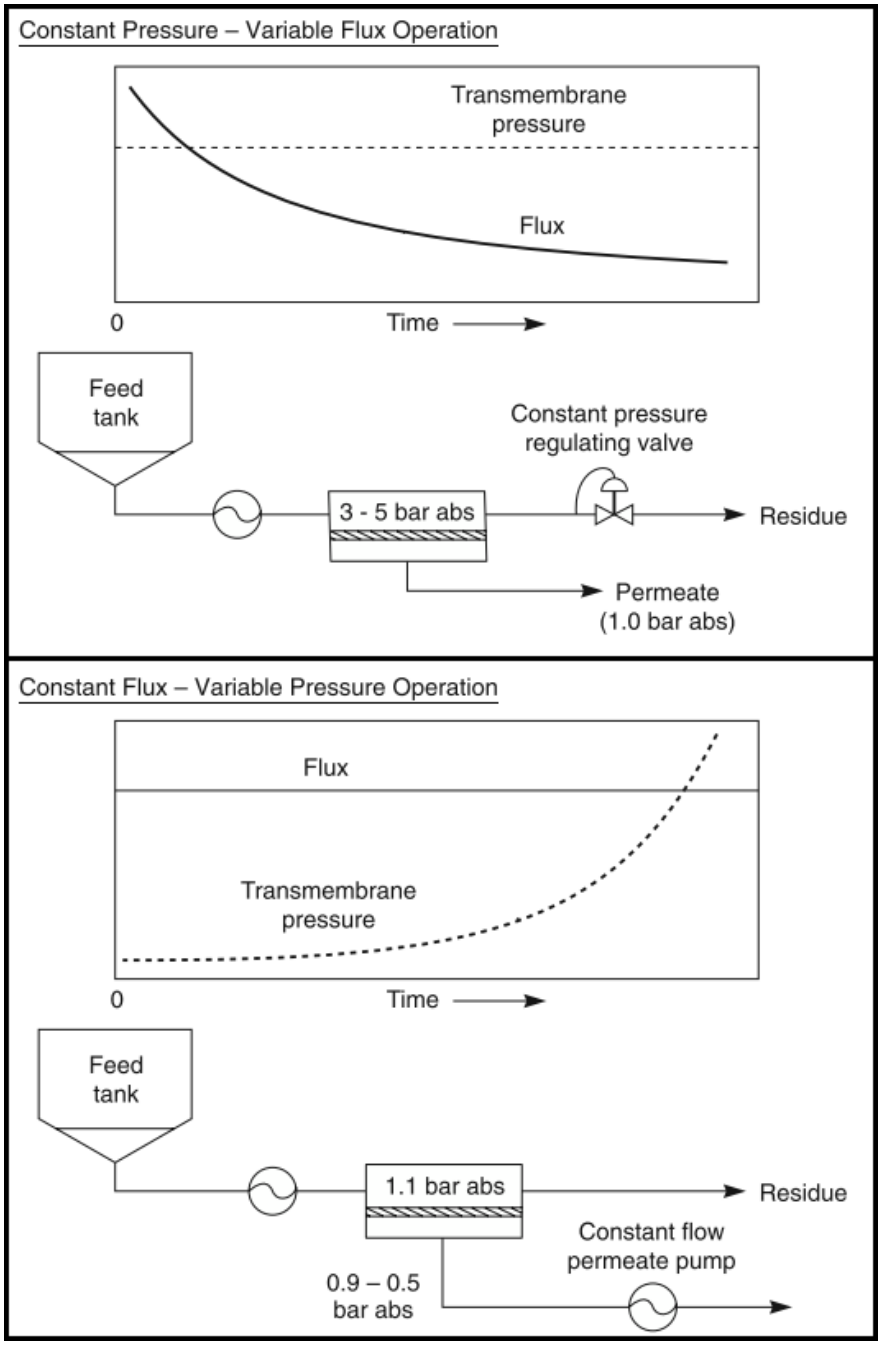


Figure 3.4: The differences between constant-pressure (above) and constant-flux (below) on configuration and evolution through time (Baker, 2012, chapter 6) [9].

4

Ultrafiltration (UF)

Ultrafiltration, as already said, belongs to the class of pressure-driven processes. Although this section will be focused on ultrafiltration, and in particular to water treatments, most of the concepts apply also to the other methods.

Note that ultrafiltration cannot be used to treat solutions but only to separate a heterogeneous system, i.e., dispersions. By employing the membrane with the right characteristics, ultrafiltration can be used to retain particles, nanoparticles, droplets, or nanodroplets dispersed in the solvent.

4.1 Models classification

To manage and optimize the flux we are interested in a function that relates the flux to the operating pressure. Over the years, many models have attempted to achieve so with varying levels of correctness and generality. Given the complexity of the topic, the vast number of factors, and the many different phenomena coming into play, a complete theory is still missing. Moreover, errors, misunderstandings, and wrong assumptions can be found in many articles and books, including famous ones, along with a non-unique nomenclature. In this thesis, notations and definitions used are based on the *de-facto* standards of the most notable works, and the unit measures have also been included (often they are omitted leading again to misinterpretations).

In general, flux-prediction models can be classified in:

- **phenomenological**: are based on filtration theory such as gel-polarization, osmotic pressure, resistance-in-series, and fouling models.

- **non-phenomenological**: are based on statistical tools such as artificial neural networks, data mining, computational models of system dynamics, and principal component analysis (PCA).

A notable work is that of Quezada et al. who, in 2021, reviewed and compared many models for permeate flux prediction in ultrafiltration. They tested the selected models using data coming from the filtration of three fruit juices (bergamot, kiwi, and pomegranate), processed in a cross-flow system for 10 hours [41]. In general, they highlighted that non-phenomenological are in general more precise, having an R-squared $> 97\%$ (while phenomenological only $> 75\%$) and lower error rates. So, from a practical point of view,

the statistical methods are better estimators of the flux but less interpretable, while phenomenological models are still useful for understanding the underlying physical and mechanistic principles.

In the same work, Quezada et al., also provide a finer classification of the phenomenological in four types, that, plus the non-phenomenological identifies in total 5 different main approaches. In [27], Field and Wu have been critical of this secondary distinction and, thus, here is omitted.

4.2 Flux vs TMP: critical, threshold and limiting flux

To enable an in-depth study of how flux evolves through time, is necessary to understand in general how the TMP can affect the flux. A neophyte reader could erroneously conjecture a linear relationship between flux and TMP: as pressure increases, one would expect a proportional increase in flux. In reality, as visible in picture 4.1, the function $J = f(TMP)$ is non-linear, and can be distinguished four main areas:

1. **non-fouling region**: at very low rates, when the $TMP < TMP_{crit}$ there is no fouling, but concentration polarization is present anyway. Unfortunately, typically, the TMP_{crit} is a very low value, and it may be the case that the number of cube meters that companies have to deal with daily can't be accomplished at such rates. Moreover, if pumps are not designed to work at such little pressures, their energy consumption may not be optimal. The maximal flux achievable under this area is the so-called **critical flux** (J_{min} or J_{crit}). Note that, the flux of pure water is known as the **plateau flux** ($J_{plateau}$) because there can't be any fouling ($C_{water} = 0$) and thus doesn't exist as well a critical flux value.

2. **low-fouling region**: the fouling region can be divided into low-rate and high-rate regions, and are separated by the **threshold flux** (J_{thre}). The concept of threshold flux was introduced in 2011 by Field and Pearce [25], to identify the rates of the $TMP \in [TMP_{crit}, TMP_{thre}]$ at which fouling has a nearly constant rate, independent on the flux. In many applications, the TMP is kept in this area because there is a good trade-off between flux rate and fouling.

3. **high-fouling region**: at medium-high rates, when the $TMP \in [TMP_{thre}, TMP_{lim}]$ fouling expands more rapidly. In this region, the high fouling rates can depend on the flux rates.

4. **completely fouled region**: at very high rates, when the $TMP_{lim} \leq TMP$ the membrane is completely fouled, and the flux becomes independent from the TMP. Reaching this area is non-sense because: (i) increases the power consumption of the pumps; (ii) increases the risk of irreversible fouling and damaging the membranes; (iii) increases the cleaning time; and (iv) there is no speed-up in the liters per hour processed. The maximal flux reachable for a given concentration is called the **limiting flux** (J_{max} or J_{lim}).

4.3 Reversibility and Transitoriness

Before detailing what fouling and concentration polarization are, must be introduced the notion of reversibility. In the literature, depending on the context, the "*reversible*" property of concentration

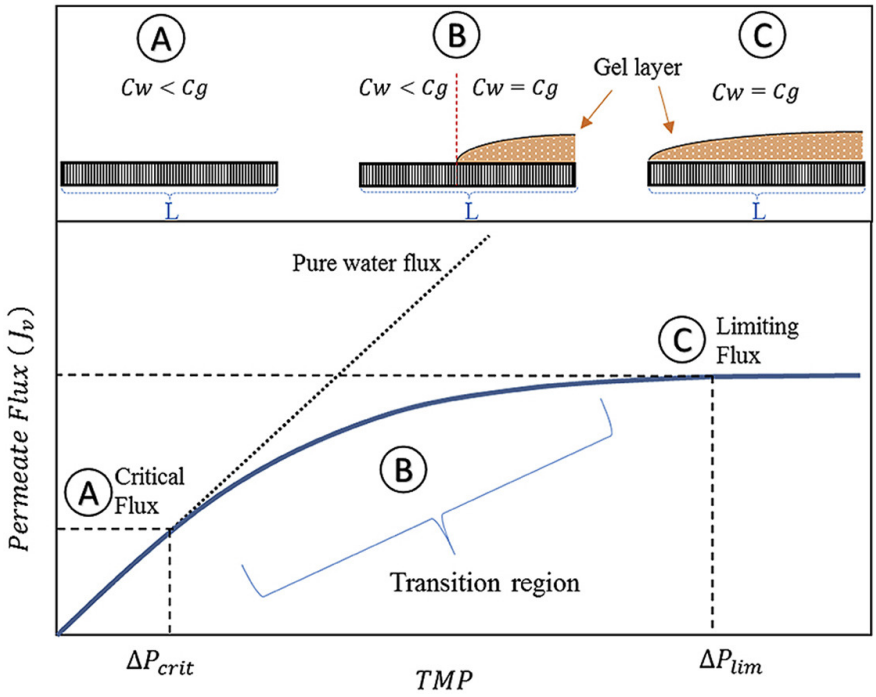


Figure 4.1: The permeate flux as a function of the TMP has three regions: (A) at low rates $TMP < TMP_{crit}$ there is no fouling; (B) at medium rates $TMP_{crit} \leq TMP < TMP_{lim}$ there is some fouling; (C) at high rates $TMP_{lim} \leq TMP$ the membrane is completely fouled (Aguirre-Montesdeoca et al., 2019) [2].

polarization and fouling can be interpreted, assumed, or implicitly defined in two ways. The most intuitive definition would define reversibility as (a) a phenomenon that can be canceled in some way, either simply by stopping the machine or maybe after a cleaning process. Therefore, an irreversible phenomenon indicates that if it occurs, the membrane is damaged, and cleaning may not fully restore its theoretical capacity. This first definition regards fouling, that can indeed damage the membrane, especially if the particle obstacle the pores in the inside on not just on the surface, as in the case of standard pore blocking, as explained in 4.4.

Meanwhile, the reversibility property of concentration polarization is defined as (b) a phenomenon that disappears as soon as the plant is turned off. So, for the latter cleaning is not necessary at all, and, to avoid misunderstandings in this thesis it will be referred to as a "*transitory*" effect. Clearly, (b) transitoriness implies (a) reversibility, but the contrary does not hold.

4.4 Fouling

Fouling refers to the deposition of suspended or dissolved substances on or in the membrane. Fouling blocks -completely or partially- the pores of the membrane, reduces its active filtration area, and thus causes the reduction of the flux. Fouling in general is not temporary, and, in some cases, may not be reversible as well.

Fouling can be classified into four different types, known as *Hermia's categorization* [34]. Hermia's fouling models have several fundamental assumptions: (i) working mode: constant pressure, dead-end filtration; (ii) membrane pores: cylindrical, parallel to each other, and uniform in diameter, and (iii)

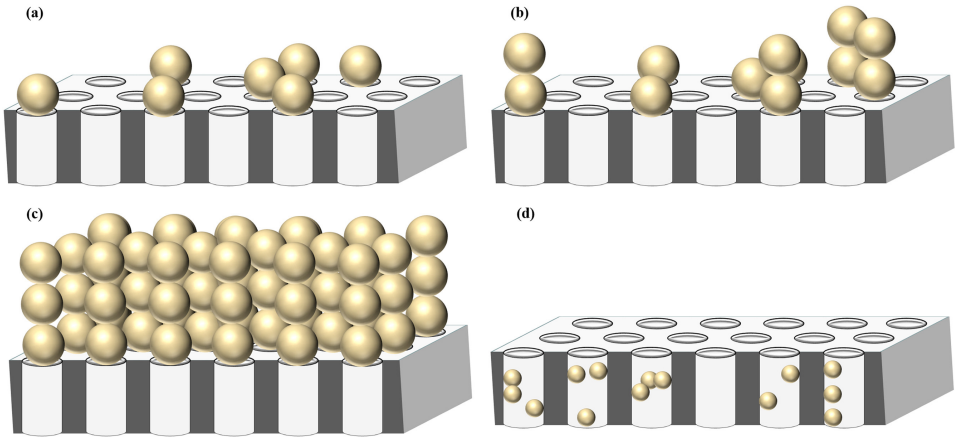


Figure 4.2: Schematic representation of Hermia's fouling mechanisms: (a) Complete pore blocking, (b) Intermediate pore blocking, (c) Cake filtration and (d) Standard pore blocking (Kirschner, 2019) [36].

foulant particles: spheres uniform and non-deformable. In reality, the classification holds even without the first assumption, but -if the working mode is the crossflow filtration- cases (a), (b), and (c) must account for the shear stress that decreases the fouling effects. The verification of one or more of the following depends on the properties of the membrane and the solution (in particular size and shape of the pores and the solute particles). The cases are shown in figure 4.2 and here pointed out, recalling Quezada et al summary [41]. Note that alongside each fouling mechanism is reported the corresponding fouling index n, which is used as a parameter in the fouling model in section 4.6.5.

(a) **Complete pore blocking ($n = 2$)** assumes that fouling occurs only on the membrane's surface. The foulant particles can deposit onto unobstructed surface areas but do not deposit on top of each other. In general, this case is considered reversible, thus after a cleaning session the membrane should be (almost) restored.

(b) **Intermediate pore blocking ($n = 1$)** is similar to complete pore blocking, but foulants can deposit on either the unobstructed surface or the obstructed surface as well, i.e. other foulants. Similarly to (a), this case is considered reversible. Differently from case (c), here the deposition on top of each other is supposed to create a cake with negligible height, thus the overall mass transfer resistance is considered to be a constant.

(c) **Cake filtration ($n = 0$)** In cake filtration, the foulants completely cover the membrane surface in several layers. In this case, the overall mass transfer resistance is not a constant anymore but rather increases proportionally to the cake layer thickness. If the flux becomes pressure-independent, additional increases on the pressure would cause the cake layer to become more compact, even risking to damage the membrane surface irreversibly. The cake is also known as the *gel layer*, as has concentration C_{gel} (or shortly, C_g), higher with respect to the concentration of the bulk feed C_f.

(d) **Internal (or Standard) pore blocking ($n = 1.5$)** is different from the cases above because assumes that blocking occurs only inside the pores, thus reducing the pore diameter. This case could irreversibly damage the membrane if the obstructed pores cannot be cleaned in any way.

4.5 Concentration Polarization

Concentration Polarization (CP) refers to the accumulation of retained solutes on the membrane surface, caused by the selective transfer of the membrane; the phenomenon is reversible and immediately occurring. According to [28] concentration polarization causes four main effects:

- alteration the **physicochemical properties** of the solution within the boundary layer (e.g., viscosity),
- amplification of the **osmotic pressure** difference $\Delta\pi$ on the two sides of the membrane, reducing the effect of the operating pressure,
- increment of the membrane's concentration C_m, thus indirectly speeding up the **fouling** process,
- increases the probability of **gelation** at sufficiently high surface concentrations.

CP is one of the reasons that causes the flux decline, but, **only for the first few minutes**, can be considered the major responsible. In fact, after the initial short period, it is just one of the causes but for sure not the primary. Note that, differently from fouling, CP is not only *reversible*, but also *transitory*, i.e., whenever the driving force is removed and permeation ceases, the phenomenon disappears.

To summarize, I quote the statement made by Field et al. [27]: "*Concentration Polarization is distinguishable from fouling in at least two ways: (1) the state of the molecules involved (in solution for concentration polarization, although no longer in solution for fouling); and (2) by the timescale, normally less than a minute for concentration polarization, although generally at least two or more orders of magnitude more for fouling. Thus the phenomenon of flux decline occurring over a timescale of tens of minutes should not be attributed to concentration polarization establishing itself*".

4.6 Modeling Flux

The main references are the books of Baker [9]; Ho, Sirkar, [35]; Scott, Hughes [45] and the articles of Cui, Jiang, Field [21]; Quezada et al. [41], and the recent "*Permeate Flux in Ultrafiltration Processes - Understandings and Misunderstandings*" of Field, Wu [27], which, as a computer scientist, helped me a lot in understanding and clarifying many doubts that had crossed my mind.

One of the first formulas to model the flux dates back to 1985 and was designed by Belfort and Nagata [13]. It states that -assuming a constant density over the bulk- the solute transported to the membrane surface by the convective flow of the permeate is balanced by the back diffusion to the bulk:

$$J(C - C_p) = -D\frac{dC}{dx} \tag{4.1}$$

where D is the diffusion coefficient of the solute, C_p is the permeate concentration, and C is the concentration at a given point within the concentration boundary layer of thickness δ.

In the remaining part of the section, will be listed and commented the most known phenomenological models. Note: since in many applications $C_p \approx 0$, it is common to see a variation of the formula without C_p. This consideration applies not only to eq. 4.1, but also for the ones derived from it such as film model, gel model, etc.

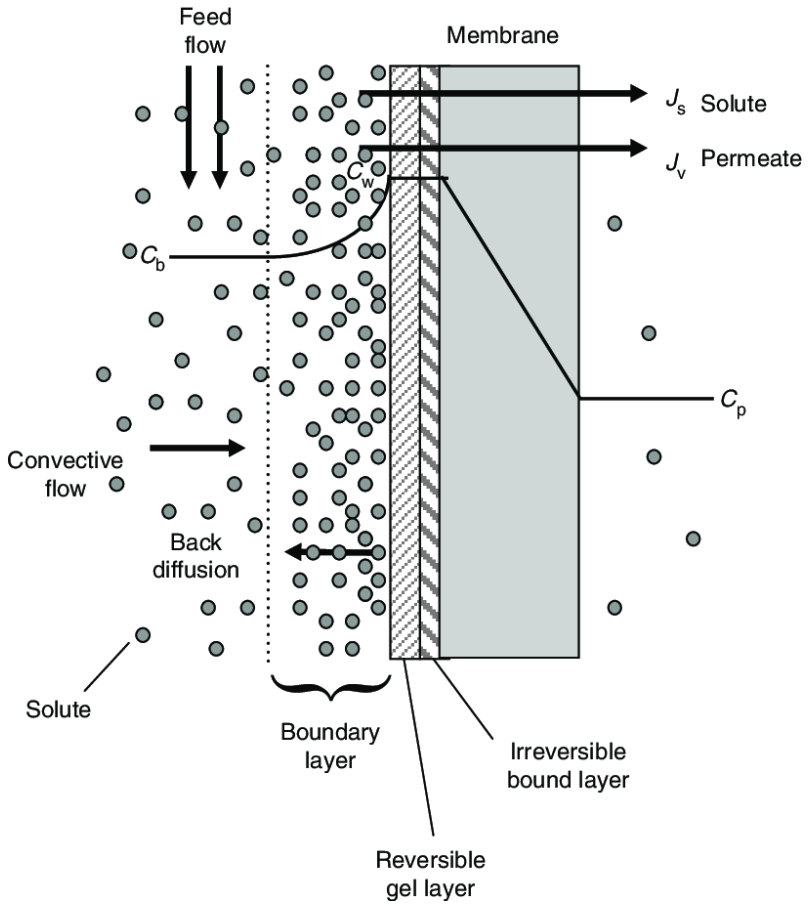


Figure 4.3: A schematic representation of concentration polarization and fouling at the membrane surface. (Goosen et al, 2004) [29].

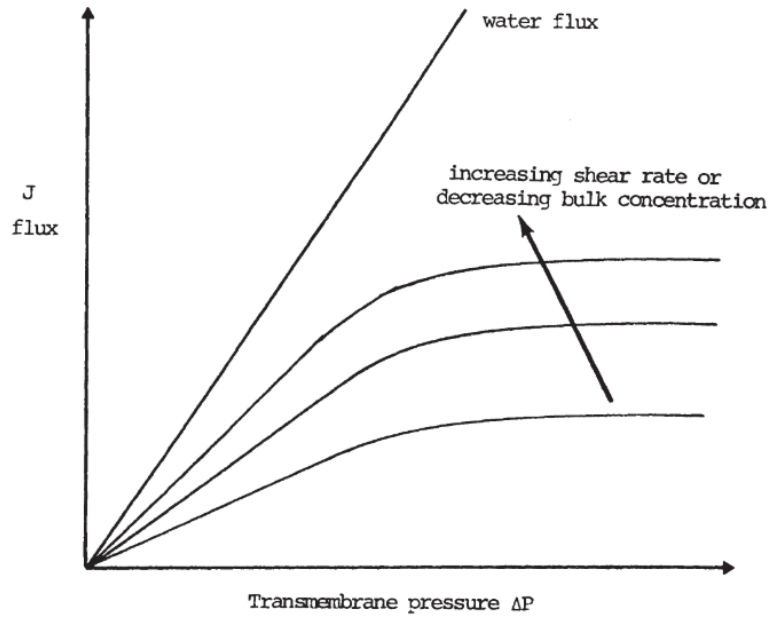


Figure 4.4: Dependence of flux on feed solute concentration: general behaviour for ultrafiltration with low viscosity feed (Scott, Hughes, 1996, chapter 4) [45].

4.6.1 Film Model

By integrating equation 4.1 over δ the film model formula is obtained:

$$J = k \ln \frac{C_m - C_p}{C_f - C_p} \; ; \;\; k = \frac{D}{\delta} \tag{4.2}$$

where C_m, C_f are the concentration on the membrane surface and that of the bulk feed respectively. The ratio D/δ is considered to be equal to k, the mass transfer coefficient measured at zero flux. This formula helped in understanding the underlying process and shows that the flux depends log-linearly on the concentrations.

4.6.2 Gel Model

The goal of the gel model is to predict the limiting flux J_{lim}, the maximal flux reachable for a given concentration. Similar to the film model, the gel model also starts from equation 4.1. In addition, erroneously assumes that the gel concentration is constant $C_{gel} = C_m$. In reality, after the measurement of various samples, was shown that C_{gel} increases with greater bulk concentrations and lower feed velocity. More importantly, the level of concentration at which a solution should gel doesn't always match the gel model predictions. For these reasons, nowadays the gel model is considered **wrong** and must be avoided.

$$J_{lim} = k \ln \frac{C_{gel} - C_p}{C_f - C_p} \; ; \;\; k = \frac{D}{\delta} \tag{4.3}$$

4.6.3 Osmotic Pressure Model

The osmotic pressure model is based on three equations:

- eq. 4.2 links the flux to the concentration, and is the one of the film model.
- eq. 4.4 associates the concentration to the osmotic pressure, and is necessary to estimate the latter. In fact, the osmotic pressure of many macromolecular solutions can be expressed through a virial expansion.
- eq. 4.5 relates together flux, operating pressure, and osmotic pressure. Note that the osmotic pressure difference is equal to the differential value across the membrane: $\Delta\pi = \pi_m - \pi_p$.

The osmotic pressure model has shown great effectiveness for many solutions. Still, this model presents an issue: is not capable of predicting the decrease in flux that happens when the TMP is high.

To address this issue has been proposed to update the value of the mass transfer coefficient k. Being the concentration on the membrane side high ($C_m > C_p$), this causes a lower permeability and, thus, greater viscosity and a reduced mass transfer coefficient k. As shown in eq. 4.6, k is defined as k_0, the mass transfer coefficient in the absence of polarization (i.e. under isoviscous conditions), times a correction factor defined as the ratio between the viscosity on the two sides, raised to some constant z. According to many works, such as [28], z is ≈ 0.14, which is the Sieder-Tate value.

$$J = k \ln \frac{C_m - C_p}{C_f - C_p} \; ; \;\; k = \frac{D}{\delta} \tag{4.2}$$

$$\pi = A_1 C + A_2 C^2 + A_3 C^3 \tag{4.4}$$

$$J = \frac{TMP - \Delta\pi}{\mu_p R_m} \tag{4.5}$$

$$k = k_0 \left(\frac{\mu_b}{\mu_m} \right)^z \tag{4.6}$$

4.6.4 Resistance-in-Series Model

The resistance-in-series model can be seen as a direct application of Darcy's law, where the flux is controlled by several resistances in series. By looking at the first equation, the osmotic pressure does not affect anymore the operating pressure, but the effects of both concentration polarization and fouling have been included. As mentioned in section 4.5, one of the main effects of CP is to alter the osmotic pressure difference, indicating some sort of correlation. Effectively, has been proved that either subtracting $\Delta\pi$ (in the numerator) or adding $\mu_p * R_{cp}$ (in the denominator) results in a thermodynamically equivalent definition ([49], [37]). In other words, eq. 4.7 is a generalization of eq. 4.5 that includes also fouling.

$$J = \frac{TMP}{\mu_p R_{tot}} = \frac{TMP - \Delta\pi}{\mu_p (R_m + R_{foul})} \; ; \;\; R_{tot} = (R_m + R_{foul} + R_{cp}) \tag{4.7}$$

4.6.5 Hermia's Fouling Model

Limiting to **dead-end** and to constant-pressure modes, Hermia was the first person to (i) give a physical derivation of the so-called intermediate blocking law and (ii) unify the four blocking filtration laws for porous media (as defined in section 4.4) into a single equation [34]. Although effective, the formula seems counter-intuitive: *how should time be double differentiated with respect to the volume?* To answer this question, one should consider that (i) what is differentiated is the time from the start of the filtration, (ii) dt/dV_p is the reciprocal of the permeate flow $Q_p = A * J \; [L/h]$ (details of this relationship at section 6.1). Given these considerations, later on, equation 4.8 was rephrased as eq. 4.9, which has a much simpler mechanistic interpretation. Note that being this a dead-end constant-pressure formula there isn't a limiting flux value blocking the derivative, hence, the flux will continue decreasing until 0.

$$\frac{d^2t}{d^2V_p} = k_n \left(\frac{dt}{dV_p} \right)^n \tag{4.8}$$

$$\frac{dJ}{dt} = -\frac{k_n}{A}(AJ)^{3-n} \tag{4.9}$$

where V_p is the permeate volume, n is the index of the corresponding fouling mechanism (either $0, 1, 1.5, 2$), k_n is a constant depending on the latter, and A is the membrane area.

4.6.6 The Cross-Flow extension of Hermia's Fouling Model

In 1995, Field et al. extended Hermia's work in the cross-flow filtration mode [26], again assuming a constant-pressure mode. Recalling the fouling mechanisms outlined in section 4.4, three out of four impact at the membrane surface, and so their effect is mitigated by the shear stress. The standard pore-blocking mechanism instead, obstructs the pores internally, producing the same effects on both working modes. Thus, Field et al. work regards partial and complete pore blocking, and cake filtration. The main difference is the inclusion of J_{min} which causes two effects. First, if the initial flux $J_0 >> J_{min}$, their difference will have a huge impact on the derivative, making it drop fast in the first minutes. As time goes on, such difference will decrease and the flux will reduce more and more slowly towards J_{lim}. Secondly, once the flux reaches its steady state, i.e., $J = J_{min}$ (or, more precisely, $J - J_{min} < \epsilon$), it stops decreasing ($J' = 0$) at $J = J_{lim} > 0$ (while in dead-end it stops at 0). Of course, for the internal fouling case $J_{min} = 0$.

$$\frac{dJ}{dt} = -\,k_n\,J^{2-n}\,(J - J_{min}) \text{ with } J \geq J_{min} \tag{4.10}$$

4.6.7 Integral Method

The above-defined equation can effectively explain the reduction in flux caused by a particular fouling mechanism. Despite that, it is possible that during the filtration process, two subsequent fouling mechanisms act. In such a case must be distinguished a **first** (or initial) and a **second** (or following) fouling phase (or even more than two). Hence, fitting k_n to the data would produce an erroneous model using any of the fouling mechanisms. Instead, the data should be split into two phases and the parameters should be estimated for each. To achieve this purpose, in 2021, the integral method was introduced by Wu [50].

Briefly, first of all, she substituted n with the four possibilities and integrated each equation, which will result in the following form: $f_n(J, J_0)/t = k_n(v/t) - k_n J_{lim}$ where $v = V_p/A$ is the specific volume. To compute the latter for each fouling mechanism is necessary to know J_0 (the initial flux at time $t = 0$) and J_{lim} (the steady-state flux). By using ARIMA models (Autoregressive Integrated Moving Average), a statistical model for time series analysis, she provides a way to estimate both. Then, by substituting $y = f_n(J, J_0)/t$, $m = k_n$, $x = v/t$, $q = -k_n J_{lim}$ the equation resulting would be $y = mx+q$, which means that all of the four predicted models are linear w.r.t. to v/t. Finally, the experimental data is plotted against the predictions. In the case where one of the mechanisms fits the data with a straight line, we're done and we've discovered the parameters. Otherwise, an inflection point will probably become apparent. In the last-mentioned case, we find ourselves in the two-phase scenario, and we just need to partition the data -using that point as a time delimiter- and repeat the process for both stages. We recommend the reader consult the original article for further details and examples.

5

Methodology

First, this chapter describes the physical plant used to collect the experimental data. Then, the chapter contains the methodologies used to conduct the experiments and to clean, process, and analyze the raw data.

5.1 Physical Plant

The plant has the following components:

- **feed tank**: has a capacity of 150 liters has a conical trunk on the bottom and is made of AISI 304 steel.
- **pre-filter**: is made of AISI 304 steel that can store one cartridge with DOE endcap type and height of 20".
- **high-pressure pump** is made of AISI 304 steel, centrifugal with open impeller, complete with an electric motor triphase 400 [VAC], 50 [Hz], 0.75 [kW].
- **circulation pump**: is made of AISI 316 steel, centrifugal with closed impeller, complete with an electric motor triphase 400 [VAC], 50 [Hz], 1.50 [kW].
- **membrane housing**: is made of AISI 316 steel, handles up to 1000 [kPa], and can store up to 3 membranes that can work in parallel.
- **membrane**: model COM0251178, produced by LiqTech, made of ceramic with 30 channels, each having a diameter of 3 millimeters, a global diameter of 25 millimeters, a length of 1.178 meters, a filtration area of 0.33 squared meters and pores of size 60 nanometers. Most of the ink droplets in the wastewater have size $> 60\,[nm]$, and, as the experimental results confirm, the membranes have been effective in filtrating the wastewater. According to the membranes' classifications given in section 3.3, the used membranes are anisotropic, tubular, porous, and inorganic.
- **control unit**: model MT8073iE, produced by Weintek, is equipped with a PLC able to datalog and plot trends of the gathered data. The export of the logs can be either done via USB or

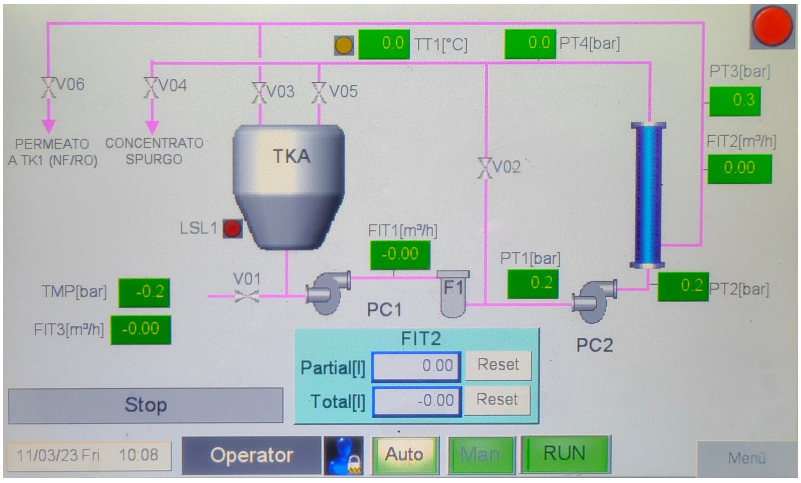


Figure 5.1: A snapshot of the control unit.

Ethernet connection, the latter also allows remote control. A snapshot of the control unit is shown in picture 5.1.

The pre-filter is included to avoid large colloids reaching, clogging, and damaging the membrane. It has been placed after the pressure pump and before the circulation pump. During the experiments, three membranes in parallel were used inside the membrane housing.

The plant has the following sensors installed: 1 level sensor (tank capacity), 3 pressure transmitters (pre-filter outlet, membrane inlet, membrane outlet), 2 flow transmitters (feed and retentate), and 1 temperature transmitter. The data from all the sensors is collected at a fixed interval of 1 minute. For safety reasons, the operating temperature must range in $[5, 50]$ °C and similarly the operating pressure in $[1000, 4500]$ kPa.

5.1.1 Plant's Workflow

The plant's workflow is presented in figure 5.2 and here detailed. (1) Initially, the feed tank is filled with the feed bulk to be purified. (2) To reach the membrane, the feed is first pushed by the pressure pump, which increases the pressure and thus the flow of the water. (3) Then, the feed encounters the pre-filter which stops large colloids from reaching and damaging the membrane. This step is not included in the model, we can assume that the variations in pressure, flow, and concentration are so low that can be ignored. (4) Then, the feed reaches we circulation pump which again increases the pressure and the flow. (5) Finally, the feed arrives at the membrane surface. (6a) Here, only the particles smaller than the pore size can go through the membrane and become the so-called *permeate*. Note that the amount of colloids that can go through does not depend only on the pore sizes, but also on other factors such as the transmembrane pressure, fouling, and concentration polarization. In general, the permeate (in image 5.1 represented by the arrow labelled with "permeato" at that end) is stored aside. Instead, during our experiments, we closed valve 06 (see again picture 5.1) not allowing the permeate to leave the cycle and forcing it to go back to the feed tank (valve 06 was open). (6b) Colloids and larger particles that do not make it through the membrane's pores and flow through the other pipe, the one labelled with "concentrato spurgo". Differently, to increase the shear stress on the membrane and improve the concentration factor, the retentate is normally introduced again in the feed, as we did.

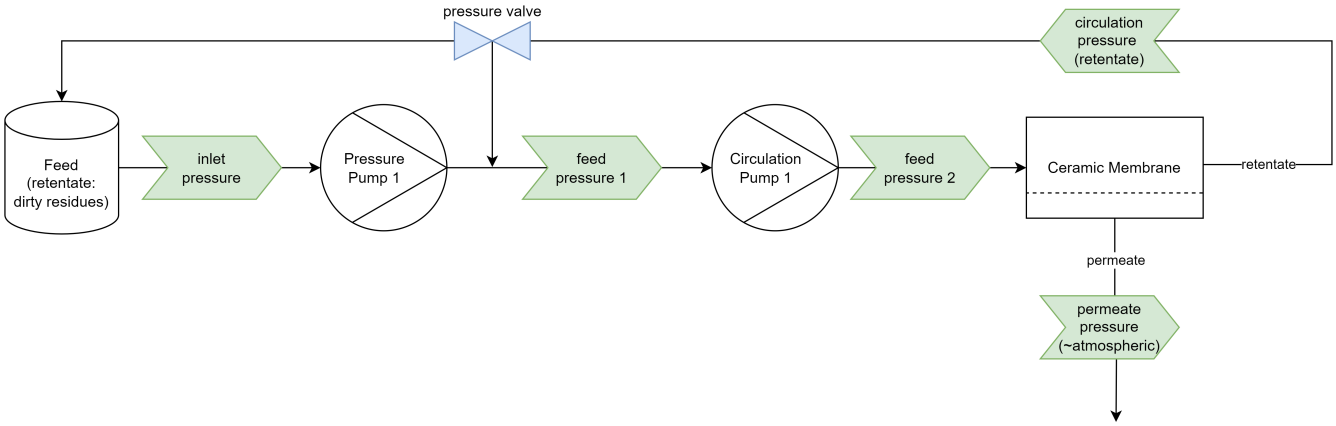


Figure 5.2: A schema of the plant workflow.

Figure 5.3: A photo of myself holding one of the membrane we used, on the background, the plant.

5.2 Methodology

The experiments have been conducted in (only) two days, the 8^{th} and 9^{th} of November 2023. On the third day, a problem occurred with the pressure pump, we couldn't use the plant anymore and had to deliver it to a repair company. For this reason, our dataset is much smaller than we expected and we plan to include new experimental data soon, possibly with real-industrial applications. Nevertheless, the dataset is sufficient to have an idea of how the process can be mathematically modeled and, with it, generate simulations not too far from reality.

Note that, to speed up the process, during all experiments both the permeate and the retentate loop back to the feed tank. This allows to obtain infinitely many samples from a finite amount of bulk feed. Clearly, the drawback is that the feed concentration instead of increasing over time (by storing aside the permeate, thus reducing the volume) actually slightly decreases: the amount of volume is constant but, since some of the foulants get stacked on the membrane surface, the bulk feed concentration indeed decreases. So, the data gathered are sufficient for a first initial modelization of the flux, but, in a real setting, the rate at which the flux decreases would be faster.

On the first day, we collected the data only by purifying clear water, to define the baseline capabilities of the plant at different TMPs. In decreasing order, we set the TMP to approximately 396, 377, 342, 285, 245, 178, 142 $[kPa]$ and let the plant work for at least 20 minutes for each TMP value. The TMPs reported here are the medians of the values registered during the process. The analysis of the clear water data is given in section 5.2.1.

On the next day, we collected the data by purifying oily water, received as a sample of wastewater from an industrial company. To run the plant with different concentration, we diluted the oily sample with tap water. We obtained and tested three levels of concentration which are: 0.4850, 1.2969, and 1.7080 $[g/L]$. For each concentration, we run the plant with many transmembrane pressure values. Each single pair $\langle concentration, TMP \rangle$ denotes a *series* in which the discrete variables are fixed. Unfortunately, we had too much hurry in changing the TMP, and for most of the intervals, we did not wait a minimum of 20 minutes to possibly see the minimum flux reached. For some intervals, we did not even wait 5 minutes, thus resulting in less than 5 samples for some couples $\langle concentration, TMP \rangle$ making those intervals basically useless. Then, as already said, on the third day we started the plant but the pump wasn't working so, we couldn't redo the experiments more calmly.

5.2.1 Baseline data: the flux of pure (tap) water

As first task, we studied the flux of clear water, checking whether if the membrane resistance R_m is indeed constant, and obtaining such value. Since there is no cooling system, due to the working pumps, the temperature rises at a rate of almost $1.2\,°C$ every 10 minutes (a detailed analysis of the temperature in section 6.5). The temperature is a key factor in viscosity. If the temperature is constant, the viscosity on the permeate side can assumed to be steady because the pressure and the concentration have very low variance. But, if the temperature rises, the viscosity decreases. As shown in eq. 4.7, the viscosity is inversely proportional to the flux. For this reason, even at a constant TMP rate, if there is neither fouling nor concentration polarization, an increase in the temperature leads to an increase in the flux. A more interesting plot would show how the flux evolves at a constant temperature. In order to do that,

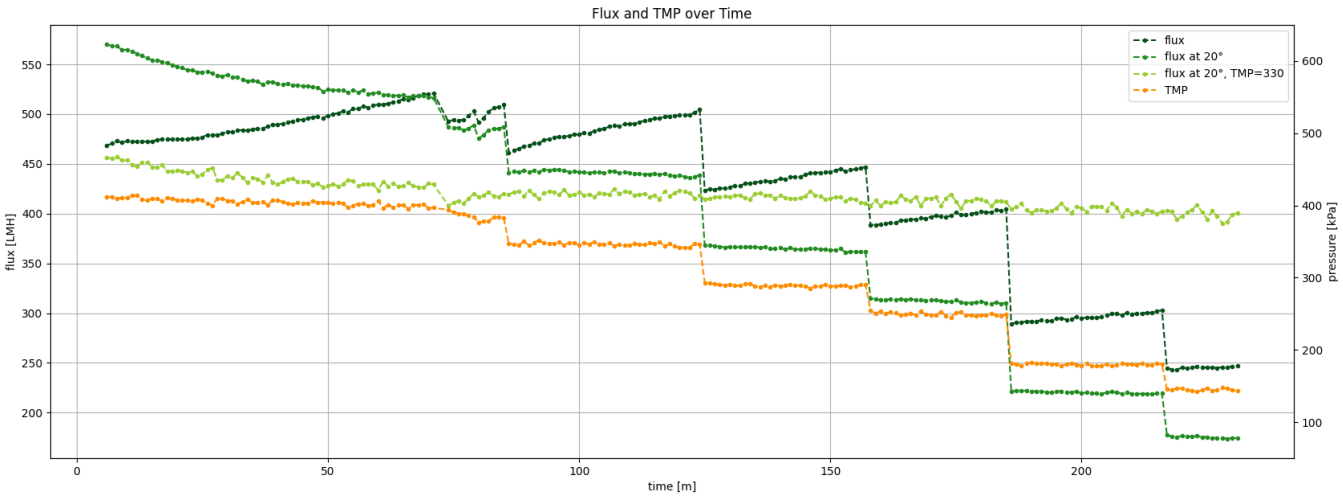


Figure 5.4: The evolution of the flux of clear (tap) water through time, at different pressures and with a rising temperature. The flux normalized only by a fixed temperature of 20° $[C]$ shows that the real flux increases because the temperature increases. The flux normalized also by a constant TMP of 330 $[kPa]$ shows that, in the long run, even when the feed is made of clear water, the flux is not perfectly stable rather is slowly decreasing.

we rescaled the actual flux using constant viscosity, assuming a constant temperature of 20 $°C$. Note that, the temperature can also affect the resistances caused by the membrane, fouling, and concentration polarization. But differently, from the viscosity, how the temperature modifies the latter is unknown. Thus, to be precise, the flux is normalized w.r.t. a constant viscosity but not perfectly w.r.t. a constant temperature. Yet, for the sake of simplicity, we have approximated the flux w.r.t. a constant temperature considering only the effects on the viscosity, with the following formula: $J_{20°} = J * (\mu_p/\mu_{p,20°})$, where $\mu_{p,20°}$ is the clear water viscosity at 20°. We also computed a normalized flux assuming both the temperature and the TMP constant, calculated as $J_{20°,TMP=k} = J_{20°} * (k/TMP)$, where TMP is the real value and w.l.o.g. $k = 330\,[kPa]$ is the constant value chosen. $J_{20°,TMP=k}$ is useful to show the non-linear dependency between the flux and the TMP: initially, big increments on the TMP cause big increments in the flux as well, but due to the flux, as time goes on, the same increments on the TMP generate smaller and smaller increments on the flux.

Figure 5.4 shows the original flux J, the flux normalized with a constant temperature $J_{20°}$, and the flux normalized with both temperature and TMP constant $J_{20°,TMP=k}$. If the bulk feed is just clear water, one expects $J_{20°,TMP=k}$ to be constant over time (no fouling). Actually, $J_{20°,TMP=k}$ starts at 456 $[LMH]$, then, in the first hour, decreases to 426 $[LMH]$, and after $3h:45min$ reaches 400 $[LMH]$. Thus, there is a difference of 30 $[LMH]$ after one hour, and a total of 56 $[LMH]$: such differences should not be ignored. If we use a linear model to predict $J_{20°,TMP=k}$ using the time in minutes as explanatory variable, the resulting coefficient is -0.23 (with a `p-value` $<$ 1e$-$95) which means that after every minute $J_{20°,TMP=k}$ decreases of 0.23 $[LMH]$.

To try to explain why the flux decreases, one could note that, in the first hour, the TMP decreases as well. Even though no valves were touched, the TMP decreases from 412 $[kPa]$ to 396 $[kPa]$. Anyway, the slight decrease of the TMP cannot explain alone the much greater decrease of the flux (also because, later on, at lower values, the TMP is indeed constant).

We conjecture that the main reason that causes the decrease of the flux over time could be a

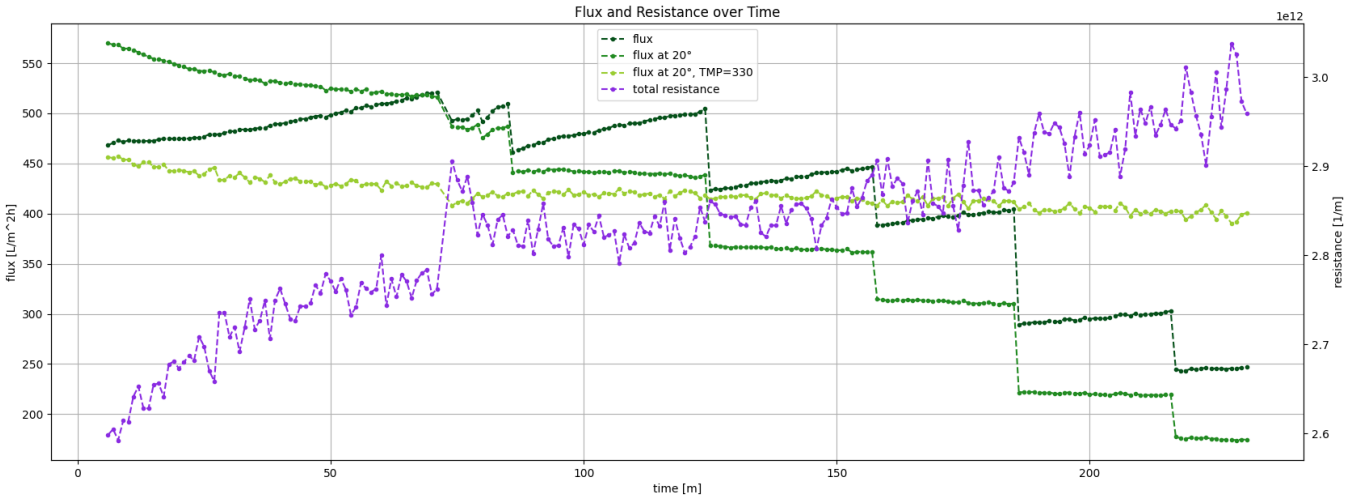


Figure 5.5: Similarly to figure 5.4, are shown the same three fluxes ($J, J_{20^\circ}, J_{20^\circ,TMP=k}$) of clear (tap) water through time. Here, the secondary axis shows the total resistance encountered by the feed at the membrane. The resistance has a growing trend, which could indicate the presence of fouling.

little bit of fouling. Considering again eq. 4.7 : to compensate for a decrease in the flux -having the viscosity and the TMP fixed-, there must be an increase in the total resistance. The membrane resistance is also a constant, or, at worst, its permeability should increase (rather than decrease) at higher temperatures. Concentration polarization can assumed to be zero (ultrafiltration membranes are not capable of retaining salts, especially when they are almost clean). Thus, the only factor remaining is fouling. Moreover, the line chart of the total resistance R_{tot}, calculated by swapping J and R_{tot} in equation 4.7, supports our thesis: clearly, it's always increasing, as visible from fig. 5.5. An interesting fact is that even in the final part of the chart, after more than 3 hours and with a $TMP < 200\,[kPa]$ (low), the resistance keeps increasing. To conclude, we think that tap water may contain small amounts of particles that get stacked on the membrane surface and cause a slight decrease in the flux over time.

This preliminary analysis has no serious implications in the modelization part, rather is a reminder that making decisions based on wrong assumptions not experimentally verified could have catastrophic implications. Moreover, it shows that even though the flux of clear water seems stable, it decreases very slowly and could take several hours to reach the fix-point. As we will see in the sections, the fouling process is much faster when the bulk feed has higher concentrations.

5.2.2 Filtration data: the flux of oily water

As visible in picture 5.6, there are 3 macro areas that correspond to the three different initial concentrations. Then, for each concentration, we tried various TMPs. One can note, according to the literature reviewed in section 4.2, different behaviours in the flux, depending on the TMP rate.

The presence of fouling here is much stronger: if initially, at low TMPs around $150\,[kPa]$ the flux decreases to $120\,[LMH]$ (and seems almost stable), then, at the same TMP rate, after the first round of increases, the membrane is fouled and the flux is stable around $73\,[LMH]$. An interesting fact is that at low TMPs, the increment of the feed concentration at time $t = 290$ has no effect on the flux at all. But, as soon as it is increased, fouling continues to expand. At high TMP values, of $435\,[kPa]$, we can see the flux's peaks: for the first concentration of $225\,[LMH]$, for the second of $202\,[LMH]$, and

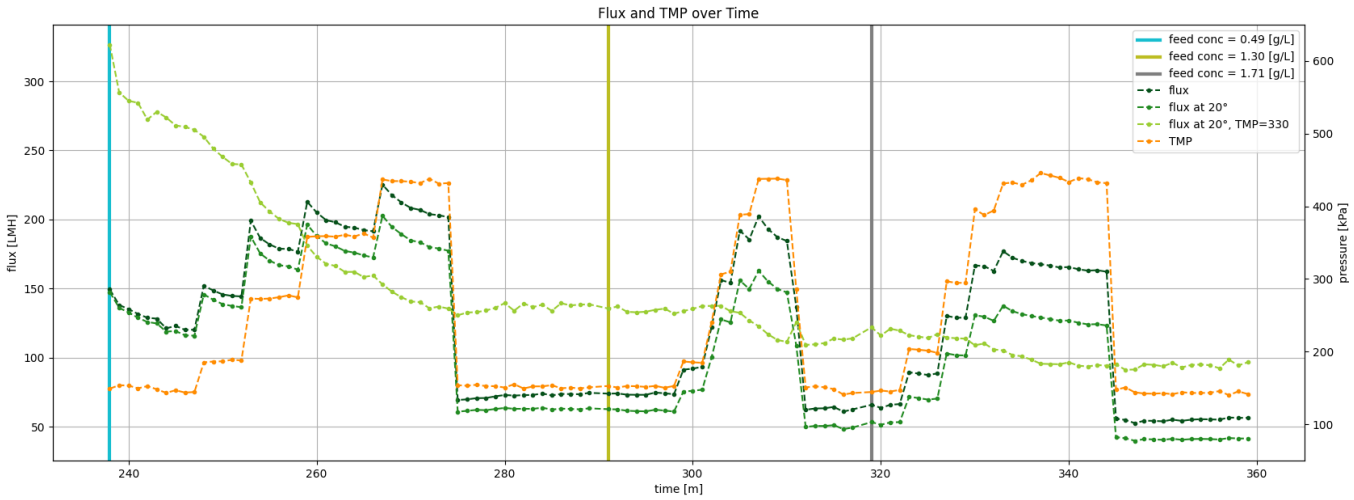


Figure 5.6: The evolution of the flux of oily water through time, with three different initial concentrations, each run with many different pressures (and with a rising temperature). The actual flux J is the raw data recorded. The flux normalized by a fixed temperature, J_{20°, shows the real flux assuming a fixed temperature of $20^\circ[C]$. The flux normalized also by a constant TMP, $J_{20^\circ,TMP=330}$, shows that, as expected, once the membrane is fouled any increase in the TMP causes an expansion in the fouling layer, but $J_{20^\circ,TMP=330}$ never increases.

for the third of $177\,[LMH]$. An additional evidence of fouling is that, after the decrease at a low TMP level, the flux is stable at around $63\,[LMH]$, thus there is a loss of $10\,[LMH]$ again the previous values of $73\,[LMH]$ at the same TMP. Again, as the concentration is further increased, no effects on the flux are visible at these low TMP levels. The effects are instead visible at higher TMP values: on the third area the flux peaks only at $177\,[LMH]$. Finally, as the TMP is lowered again, the flux stays around $55\,[LMH]$. Actually, in the last case, the TMP is a bit smaller than before ($143\,[kPa]$), but in any case, as the normalized flux $J_{20^\circ,TMP=300}$ shows, the efficiency is at its lowest.

Finding a good estimation for the TMP_{crit} -the maximum TMP where there is no fouling at all-, for the TMP_{thre} -the maximum TMP where the fouling rate is low and independent of the flux-, and for the TMP_{lim} -the minimum TMP where the flux becomes independent from the TMP- (detailed definitions at section 4.2) is out of the scope of this preliminary work. Yet, since the flux is quite stable at $150\,[kPa]$, the TMP_{thre} should be around this value, while the TMP_{crit} is probably even smaller (initially there is a decrease in the flux). Differently, at higher pressures with a $TMP > 200\,[kPa]$, the membrane gets more and more fouled, and the flux rapidly decreases at a rate of approximately $a * e^{-t}$, where a is a constant and t is time. Our maximum experimented TMP is $\approx 430 - 440\,[kPa]$. At these rates, for the third concentration at $t = 329$, the increase in the flux is low, which means that the TMP-independent area, and so TMP_{lim}, are pretty close.

Finally, figure 5.7 compares the flux of clear water (on the left) with the one of oily water (on the right). The wastewater flux is much lower, from J_{20° is visible that the increases in the TMP cause smaller and smaller increases in the flux. Moreover, note how rapidly the resistance grows, becoming up to 10 times bigger w.r.t. to the resistance of the clear water. In the superimposed chart, the clear water resistance seems almost constant because it ranges in $[2, 3]$e12 $[1/m]$, while for the oily water it spans in $[4, 14]$e12 $[1/m$.

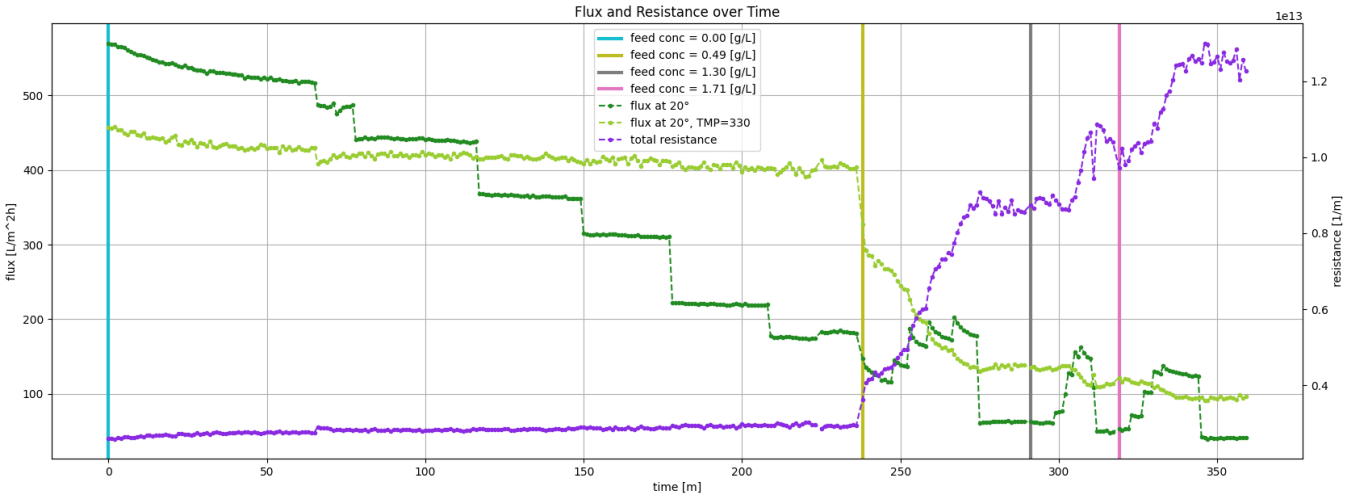


Figure 5.7: Here are shown only the normalized fluxes ($J_{20^\circ}, J_{20^\circ,TMP=k}$) of both clear and oily water through time. The secondary axis shows the total resistance encountered by the feed at the membrane. By superimposing the flux of both clear and oily water in the same scale, the resistance of clear water seems almost constant because has a range of $[2, 3]$e12 $[1/m]$, while for the oily water spans in $[4, 14]$e12. As the feed concentration is not null anymore, due to fouling, the flux normalized by the TMP drops immediately from 400 $[LMH]$ to 300 $[LMH]$, and then it continues decreasing until 100 $[LMH]$.

5.3 Augmenting dataset: forecasting with ARIMA

As already explained, lots of samples of wastewater data are too short because we changed the TMP too soon. To overcome this issue, we augmented the dataset by forecasting with ARIMA models how the flux will continue in the future. Of course, by using this method we add synthetic samples into the dataset, yet allows us to study the flux over a longer time period. ARIMA stands for AutoRegressive Integrated Moving Average and is a statistical model used to make forecasts based on old time-series data. Briefly, an ARIMA model combines 3 components: (i) the *differencing* (I) method that is applied d times to the data to make sure it is stationary; (ii) the *AutoRegressive* (AR) model, which is similar to the linear regression model but the predictors variables correspond to the last p elements of that same variable (thus *auto*regression): $y_t = \alpha + \epsilon_t + \sum_{i=1}^{p} \beta_i y_{t-i}$; (iii) the *Moving Average* (MA) model, which is similar to the AR model but the predictors are the last q forecast errors $\epsilon_{t-1}, ..., \epsilon_{t-q}$ (even though we do not observe the error values): $y_t = \alpha + \epsilon_t + \sum_{i=1}^{q} \gamma_i \epsilon_{t-i}$, with $\epsilon_t \sim N(0, 1)$, and $\alpha, \beta_i, \gamma_i$ the estimated parameters. Each couple $\langle concentration, TMP \rangle$ with at least 5 time points has been extended with the ARIMA model implemented in the Python library `statsmodels.tsa.arima`, using as parameters $(p, d, q) = (1, 1, 0)$; while the remaining series with up to 4 samples were discarded.

Given a time series, the parameters (p, d, q) should be set according to the series itself. To have a uniform algorithm that forecasts the various series, the parameters p, d, q have been kept equal for all series. In any case, for all the series there is a log of the analysis which contains the plots `PACF`, `ACF` and the function `auto_arima`; both are used to spot the best triplet for a given times series. By setting $d = 1$ we difference the series only one time: applying the differencing is necessary because the flux is not stable, and differencing one time is sufficient to make it stationary. By setting $p = 1$ we use only the precedent value to forecast the next, while having $q = 0$ implies that we do not use the moving average model at all. By visually validating the forecasts, the triplet $(p = 1, d = 1, q = 0)$ seems to be the best

choice, and, in fact, in most cases, is the triplet suggested by the `auto_arima` function. Note that, in general, an ARIMA model is not suitable for long-term forecasting. In our specific case, the forecasts only create a curve that tends to a fixpoint (the forecasted limiting flux J_{lim}) without any seasonality (like the `sin` function); thus, generating longer and longer forecasts only produce value infinitely nearer to the asymptote.

5.4 Parameters' estimation

In order to simulate the flux over time, we must be able to compute its derivative (eq. 4.10). To do so, we need first to know the flux slope k_n, the initial flux J_0, and the minimum flux J_{min}. In this section, we will see how we estimated or predicted such values.

5.4.1 Predicting J_{min}

In theory, J_{min} should not even be predicted but observed. Yet, any of our series is not long enough to see its real flux's fixpoint; thus, we have to predict J_{min}. For each series, we predicted J_{min} using the last forecasted value of the flux (after a sufficiently long sequence). Note that, the prediction *per se* is straightforward, but its accuracy depends on the goodness of the forecast, which is unknown.

5.4.2 Estimating $R_{tot}^{(0)}$

To compute the initial flux J_0, as detailed in 5.4.3, we use equation 4.7, which requires to know the transmembrane pressure, the permeate viscosity and the total resistance. The initial TMP is a parameter that the user sets, the permeate viscosity used is that of clear water, as discussed in 6.5. Thus, the only remaining unknown variable is the initial total resistance. This is a parameter that the user can set as well and depends on the membrane permeability. A precise estimation of the total resistance is a goal we haven't achieved yet. For the moment, we just set the initial total resistance to its first recorded value with the oily water sample, i.e., 3.28e12 $[1/m]$.

5.4.3 Estimating J_0

As for J_{min}, the initial flux J_0 should be observed as well. If for J_{min} we made the mistake to not waiting enough time, observing the real J_0 is a more subtle problem and was impossible for us. In fact, to know the real initial flux J_0 would be necessary to gather the sample data with a finer collection rate of either 30, 15, 5, or 1 second; but, in our plant, it was set to 1 minute and we couldn't change it. So, since the sensors collect the data every minute, we do not know how much the flux really increases from the instant in which we closed the retentate valve till the instant in which the CPU stores the sensor data. Yet, every time the TMP is changed, we always see a jump in the after a minute, and so we conjecture that the real jump is greater than the recorded one.

To estimate J_0, for the sake of simplicity, we assumed that the flux's jump is instantaneous and reaches J_0. We start from the data of a fixed couple $\langle concentration, TMP \rangle$, with its series of values for the flux $J^{(t)}, J^{(t+1)}, ..., J^{(t+n)}$ and for the total resistance $R_{tot}^{(t)}, ..., R_{tot}^{(t+n)}$. At time $t+n$, there is an increase of the TMP, and a new series identified by $\langle concentration, TMP' \rangle$, starts. To estimate J_0,

the new initial value of the flux with transmembrane pressure TMP' at time $t+n$, we add an initial fictitious time instant between the last value recorded of the previous series and the first value recorded of the following one. J_0 is then estimated using equation 4.7, with the transmembrane pressure TMP' of the following series and the last recorded resistance of the precedent series $R_{tot}^{(t+n)}$.

Note that, since (i) the increase in the flux may not be as immediate as assumed, and (ii) by increasing the TMP, the fouling and the resistance increase; this approach probably overestimates a bit the actual flux. Yet, in the Uppaal simulations we do not have the real values, and starting the new series with the resistance recorded until that moment is the most intuitive thing to do. One could ask *"When the TMP is updated, why not increase the total resistance before estimating J_0?"* The question is valid, but, as the plots of the real data show, the total resistance doesn't seem to have a big jump when the TMP is updated, only an increase in its growth rate (its derivative). Thus, adding a fictitious jump would overestimate the total resistance and would not match its behaviour in the real data.

Remark Once initialized at time $t = 0\,[min]$, the total resistance only changes according to its derivative and is never updated via a jump when the TMP is changed. Differently, the initial flux J_0 is recomputed after every discrete transition depending on the current R_{tot} and TMP.

5.4.4 Estimating J_{min}

This section could appear as a repetition of section 5.4.1 but it is not. Before, we've shown how to predict J_{min} for any series. Simply, given a series of values of the flux, we predict J_{min} by looking at its fixpoint, eventually after extending the series with the ARIMA forecast (as in our case).

The problem is that the above procedure finds J_{min} given in input the whole series. The thing is, we want to estimate J_{min} to be able to compute the flux differential equation to obtain the series itself. In other words, when we need to simulate the flux in Uppaal, we need to know *a priori* the value of J_{min}, and not deducing it as *a posterior*. So, we need a way to estimate in general J_{min}, but the problem looks like a dog chasing its tail.

To overcome this issue, we defined a linear model that takes in input only what we know in the beginning: the initial flux $J_0\,[LMH]$, the estimated $TMP\,[kPa]$, and the $R_{tot}^{(0)}$, obtained as detailed in the previous sections. We filtered the dataset including exclusively the samples of non-clear water for a simple reason: J_{min} and J_0 of clear water are almost identical.

J_0 has been included because the initial flux can give a first estimation on the value of J_{min}, smaller than J_0 but not too far from it. The following model is derived using only J_0 as predictor variable:

$$J_{min} = 15.90 + 0.672\,J_0 \;\; [L\,/\,m^2\;h] \tag{5.1}$$

As expected, it is already a good model, with `R-squared` $= 0.989$, `MAPE` $= 5.26\%$, `p-value` $<$ 5e$-$9, but a not so satisfactory `maxAPE` $= 15.39\%$. Reasoning on the outcome of the model, it says that the J_{min} can be approximated with the 67% of J_0, plus a minimum of almost $16\,[LMH]$, independently on the concentration of the feed and thus on fouling. This model resembles only how small our dataset it, because we know that fouling actually matters. Moreover, the offset of $15.90\,[LMH]$ seems artificial and indeed it is, we expect that experiments run at extremely low TMP would make this linear model wrong. Least but not last, the worst-case error even in our small dataset is already greater than 15%.

Given all the above considerations, we added to the model the transmembrane pressure and the total resistance, in order to generalize the above model, accounting also for fouling. In truth, the TMP and the R_{tot} alone do not say that much. In fact, a linear model that uses only one of the two is meaningless: both would have bad `R-squared` < 0.55 and `maxAPE` $> 60\%$. If we combine only one of the two with J_0, their contribution would be useless, with the resulting `p-values` of 0.51 for the TMP and 0.92 for the R_{tot}. Instead, if we build a model with both of them (without J_0), the model is quite solid, has an `R-squared` $= 0.977$, both `p-values` $< 1e-5$ but a not so much satisfactory `maxAPE` $= 30\%$. The question one could ask is "*Why is that?*", and the answer is simple: cross-correlation. The idea is that the TMP initially has a strong correlation with the flux (and thus with the steady point), but when fouling arises, the TMP becomes less and less effective in predicting the flux. Since the value of R_{tot} depends on fouling, combining them improves the model: when R_{tot} is low the TMP is highly relevant, but, when R_{tot} grows, the TMP becomes less informative. The final model with all the three variables is:

$$J_{min} = 52.72 + 0.428\, J_0 + 0.107\, TMP - 3.828\text{e}{-12}\, R_{tot}^{(0)} \;\; [L\,/\,m^2\;h] \tag{5.2}$$

This model is a bit harder to comprehend, the baseline is set to $52.7\,[LMH]$ to which is added the 42.8% of J_0 and the 10.7% of the TMP, and then the resistance decrements the total obtained depending on its value (and so, on fouling). The model has an `R-squared` $= 0.995$, `MAE` $= 2.80\,[LMH]$, `MAPE` $= 3.61\%$, `maxAPE` $= 9.21\%$. The `p-values` are: 0.004 for J_0, 0.036 for TMP, 0.044 for R_{tot}, thus, the last two are considered significant only at a 5% level. The good news is of course that the average error has decreased by 1.65%, and the maximum error is now below the 10%, showing a reduction of 6.2%.

However, the bad news, as a statistician might point out, is that the predictor variables should generally be independent from each other; while the latter model is highly multicollinear. The model with only J_0 is more imprecise but statistically more reliable. Specifically, the `AIC` and `BIC` are both > 63, and the `F-statistic` $= 696$. In contrast, the model with all of the three variables has both the `AIC` and the `BIC` < 60.5, and an `F-statistic` $= 388$. More importantly, the variance inflation factor (VIF), which measures multicollinearity among predictors, is concerning: $\texttt{VIF}_{J_0} = 10.3$, $\texttt{VIF}_{TMP} = 16.9$, and $\texttt{VIF}_{R_{tot}} = 4.2$. In general, a `VIF` around 4 or 5 is used as the maximum threshold, while a `VIF` ≥ 10 indicates that the multicollinearity is severe, as in this case.

Even though (i) the second model has severe multicollinearity problems, and (ii) the simpler model is not that bad; we decided to currently use anyway the second model. The reasons are two: the first is obvious, it grants an overall estimation of the flux is more precise. The second is based on a mechanistically reasoning, that supports the idea that all the factors are necessary. J_0 is the first indicator for J_{min}, but alone cannot be very precise: J_0 and J_{min} could be either pretty close when the concentration and fouling are low, while farther apart when the fouling is more intense. To account for fouling we include R_{tot} and TMP. To overcome the issue of high `VIF` values, apart from removing predictor variables from the model, alternative solutions are applying dimensionality reduction (such as PCA) or regularization (Ridge or Lasso Regression) methods. We did not explore such techniques because our training dataset is small, and before applying those transformations would be more insightful to try the model on a bigger dataset. A comparison of the two models against the forecasted value can

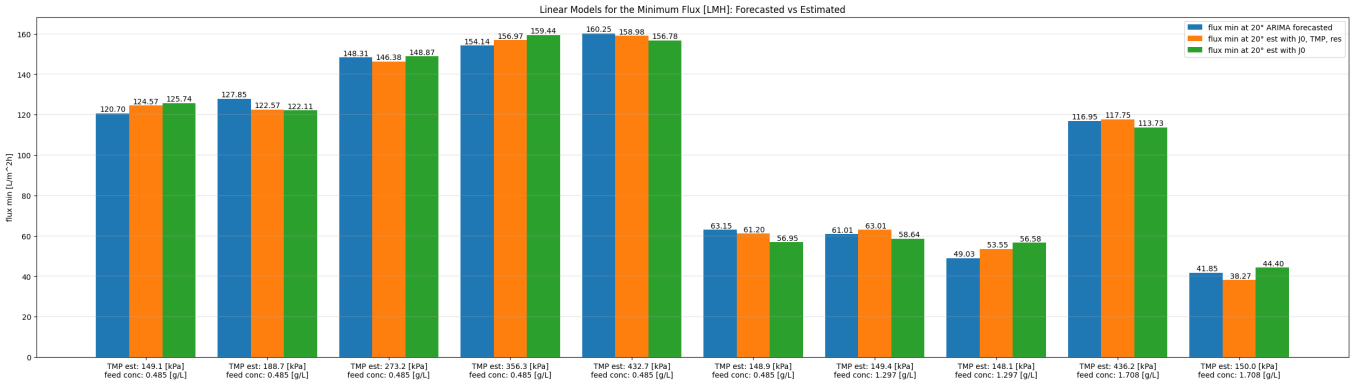


Figure 5.8: In blue is the response variable, the minimum flux J_{min}, obtained by forecasting the real data points over a longer period. In orange the estimated response variable is computed using a linear model made of the initial flux J_0, the initial total resistance, and the estimated TMP. In green the estimated response variable is computed using a linear model made only of the initial flux J_0. The orange model is on average 1.65% more precise, but its explanatory variables are highly correlated, while the green model

be viewed in picture 5.8.

5.4.5 Estimating the fouling model n and its coefficient k_n

The final missing element in the puzzle is understanding the fouling mode, which significantly impacts the differential equation 4.10 governing flux modeling. However, identifying the exact fouling model has proven challenging.

Given that ink droplets are larger than the membrane's pores, we can exclude the standard pore-blocking mechanism, as most droplets cannot enter and clog the pores. Yet, we remain uncertain about the correct fouling mechanism among the other three possibilities.

The difficulty arises from our short time series data, rendering ARIMA simulations impractical for deriving a fouling model. Nonetheless, we attempted this approach, observing that the computed k_n has almost the same behaviour across any n, apart from a multiplier factor. The unexpected result is that k_n before stabilizing to a constant value, initially has an almost linear decrease over time for the first 5-6 minutes. This suggests that k_n may not remain constant at all. Moreover, the instant at which k_n stabilizes (around 5-6 minutes) coincides with the transition from observed data to forecasted data. This prompts questions: is the change in k_n after a few minutes real, or is it a result of flawed simulations? Moreover, is could be also the case that after 5-6 minutes there is a change in the fouling model, where for example an initial phase of complete pore blocking is then followed by a subsequent phase in which a cake forms.

To address these uncertainties and gain a comprehensive understanding, two crucial steps are required: obtaining longer data series and collecting samples directly from the membrane surface. While acquiring longer data is relatively straightforward, it alone may not conclusively answer these questions. Analyzing the membrane surface is essential but considerably more time-consuming. Unfortunately, our efforts to generate longer time series were hindered by a failure in the pressure pump, preventing further data collection.

Given the membrane and feed properties, we assumed that the fouling type may be a cake filtration,

thus $n = 0$. We then conducted Uppaal simulations with various values until identifying a suitable candidate that closely matched the real data. Ultimately, we set $k_n = 5.5e - 6$. Even though we can't compute any goodness measure on these parameters, we can check how the evaluate the precision of the whole model. This task is later presented in section 9.

6

Mathematical Model

In order to have a system as realistic as possible, the model is based on a hybrid dynamical system 2.1.8: *dynamical* means that its state evolves through time, meanwhile *hybrid* implies that its state contains both discrete and continuous variables. *Discrete* variables have a constant value inside states and can be updated when changing states by the transitions, on the contrary, the *continuous* evolve inside states and are modeled by flow conditions such as differential equations. Note that, when a discrete event is verified and a transition is taken, it could alter also the values of the continuous variables, adding a point of discontinuity (a *jump*) in their timeline.

Specifically, almost all the variables are considered continuous, apart from the ones that are stationary and have low variability, which have been assumed to be discrete. The latter corresponds to all the various pressures ($P_r, P_{f_1}, P_{f_2}, TMP$) plus the flow of the retentate (Q_r). There is -for the moment- only one possible discrete event that can happen: when an operator (or the machine itself if supported) alters the pressure valve and causes the change of the pressures. In the mathematical model, the pressure valve does not exist and the controller can instantly change the pressure of the retentate (P_r), and, as a consequence, all the other discrete variables as well. Note that, changing the P_r and so the TMP also the continuous flow-related variables will be updated as well R_{tot}, J, Q_p, Q_f.

Notation Note 1 Since each variable depends on the moment in which it is measured if the time of the evaluation is not specified it is assumed to be the current moment i.e. $x = x^{(t)}\ \forall x \in Vars$.

Notation Note 2 Any derivative indicated with the apostrophe is assumed to be derived with respect to time in minutes i.e. $x' = \dfrac{d}{dt}\, x\, \left[\dfrac{\cdot}{min}\right]\ \forall x \in Vars$.

Implementation Note: any parameter found via a linear regression model was obtained using the function `OLS()`, available inside the python library `statsmodels.regression.linear_model`, which implements the least squares method. For the estimated models and parameters will be given the `p-values` alongside some classical regression metrics (computed thanks to the python library `sklearn.metrics`) such as `(Adj) R-Squared` (R-Squared and its adjusted version), `RMSE` (root mean squared error), `MAE` (mean absolute error), `MAPE` (mean absolute percentage error), `maxAE` (max absolute error), and `maxAPE` (max absolute percentage error).

6.1 Axioms and Identities

Some well-known identities and equations were used as a basis to construct the entire model.

Flux Equation The general equation that models the permeate flux in a filtration system, the one of the resistance-in-series model previously introduced. Working at a constant pressure rate, the flux will decline over time due to the concentration polarization (R_{cp}), in the first few minutes, and later on due to the fouling (R_{foul}).

$$J = \frac{TMP}{\mu_p \, R_{tot}} = \frac{TMP}{\mu_p \, (R_m + R_{foul} + R_{cp})} = \frac{TMP - \Delta\pi}{\mu_p \, (R_m + R_{foul})} \;\; [L \,/\, m^2 \; h] \tag{4.7}$$

Practical considerations: as already said, the implemented model is based on the ODEs which are solved by Uppaal. Thus, this formula is not directly included in the model but is used for two scopes: (i) derive the ODEs of the total resistance (as later shown in eq. 6.18), and (ii) to the derive the total resistance in the experimental data to validate the model (J and TMP are given, μ_p is approximated with that of pure water at pressure P_p).

Transmembrane Pressure Equation One of the most important equations in pressure-driven filtration systems: the transmembrane pressure can be approximated by averaging the pressure on the input side of the membrane (P_{f_2}) and the one on the output side (P_r), minus the permeate pressure (P_p). Since the variation of pressure is the driving force that allows the water to flow, is often represented also as ΔP.

$$TMP = \Delta P = \frac{P_{f_2} + P_r}{2} - P_p \;\; [kPa] \tag{6.1}$$

Practical Considerations In our case, the TMP is not continuous: it changes only when we open or close the pressure valve and is modeled as a discrete transition. Thus, there's no need to define an ODE for it, its update is performed while taking the transition using equation 6.1.

Flux Differential Equation in Cross-Flow Filtration (under constant pressure) The alternative definition of the flux for cross-flow filtration is defined via a differential equation, as suggested by Field et al. as an extension of Hermia's model in dead-end filtration. Note that from the experimental data the flux is already given, and its derivative can be approximated by taking the difference from two consecutive data points: $J' \approx J^{(t)} - J^{(t-1)}$. By waiting enough time ($\approx 20 - 30\, mins$), J_{min} can be extrapolated experimentally by observing that the flux converges to a fixed point. As a consequence, k_n is the only unknown variable left, and so it can be computed as well.

$$J' = -\, k_n \, J^{2-n} \, (J - J_{min}) \text{ with } J \geq J_{min} \tag{4.10}$$

where $n \in \{0, 1, 1.5, 2\}$ represents the fouling mechanisms in Hermia's categorization, J_{min} is the value of the steady flux (the fixed point where it stops decreasing), and k_n is a constant depending on n. **Practical Considerations** As above-said, the TMP is neither a constant nor continuous. This implies that we can indeed use this ODE to define the flux behaviour

over time, but, when the TMP is updated, must be updated as well the flux, using equation 4.7. The jump of the flux is assumed to take place instantly.

Relationship between Flux and Permeate Flow The following relationship is quite trivial: they represent the same information. The only difference is that the flux abstracts from the actual area of the membrane -more convenient in a mathematical study-, meanwhile, the permeate flow allows the calculation of the effective volume of the permeate per hour since it multiplies the flux with the area. The latter is instead more useful from an engineering point of view because it allows understanding the daily capacity of the plant and following reasonings.

$$Q_p = A * J \ \ [L \,/\, h] \tag{6.2}$$

Practical considerations: Q_p is used to estimate the consumption energy of the pumps and the permeate concentration.

Relationship between Permeate, Retentate, and Feed Flows Here the identity is pretty intuitive: the same amount of flow on the inlet side of the membrane is preserved on the outlet, split in permeate and retentate.

$$Q_f = Q_p + Q_r \ \ [L \,/\, h] \tag{6.3}$$

Relationship between Flow and Volume Since the volume of the permeate is the integral of the flow of the permeate, by applying the derivative to both sides we obtain the relationship.

$$V_p' = Q_p \tag{6.4}$$

Solute Balance The solute balance is a direct consequence of the law of mass conservation: the total initial amount must correspond to the final one. The mass $[g]$ of a solution is obtained by multiplying its volume V $[L]$ with its concentration C $[g \,/\, L]$. In the notation $X^{(i)}$, X stands for the variable of interest, and i for that value at time i. So, for instance, $C_r^{(i)}$ means the concentration of the retentate at time i.

$$C_f^{(0)} * V_f^{(0)} = V_r^{(t)} * C_r^{(t)} + V_p^{(t)} * C_p^{(t)} \ \ [g] \tag{6.5}$$

6.2 Evaluation Metrics

To quantify the effectiveness and the efficiency of the filtration process different metrics can be defined. *Effectiveness* metrics are used to estimate *how good the filtration process is* and typically are dimensionless percentages. *Efficiency* metrics take into consideration also the costs. The only efficiency metric here presented is the normalized energy use while all the remaining regard effectiveness.

Solute Rejection Coefficient Defines the percentage of solute not retained that has gone through the membrane, so it is a measure of effectiveness. When the process has a batch setup (not a stream) can be calculated by simply rearrangeing the solute balance (see equation 6.5). The

goal of filtration systems is to increase the concentration in the retentate and decrease it in the permeate. When the latter happens, R increases, up to 1. Note that, in general, the rejection coefficient taken under consideration is the *observed* (or *apparent*) one, which is much easier to obtain. In reality, this value is influenced by concentration polarization and the real rejection coefficient of the membrane should consider the concentration on the membrane, which can be sensibly higher than that of the feed ($C_m >> C_f$).

$$\mathbb{R} = \mathbb{R}_{obs} = \frac{C_f - C_p}{C_f} = 1 - \frac{C_p}{C_f} \tag{6.6}$$

$$\mathbb{R}_{real} = \frac{C_m - C_p}{C_m} = 1 - \frac{C_p}{C_m} \tag{6.7}$$

Volume Concentration Factor Defines the ratio between the volume of the initial feed and of the final retentate. Note that, since mass concentration grows inversely proportional to volume, the goal is to decrease the volume of the retentate, thus maximizing R. Moreover, the VCF can be used to estimate how much storage space will be needed for the filtered solution. In water treatments, the retentate typically consists of solid waste. Lower retentate volume results in cheaper discharge expenses, thus, maximizing the VCF induces a cost reduction.

$$VCF = \frac{V_f}{V_r} \tag{6.8}$$

Recovery Rate Represents the percentage of initial feed volume that has been successfully cleared. In industrial applications, the maximization of the recovery rate is one of the key factors. In fact, it is used to estimate the amount of water that can be reused.

$$\mathcal{R} = \frac{V_p}{V_f} \tag{6.9}$$

Specific Energy Consumption A typical metric to evaluate the efficiency of a system is the ratio of the energy use over the amount of permeate volume produced. Note that, the energy costs are only a part of the total costs related to an plant.

$$SEC = \frac{E_{tot}}{V_p} = \frac{E_{fp} + E_{cp}}{V_p} \quad [kWh/m^3] \tag{6.10}$$

6.3 Constants

Inlet Pressure The pressure of the water entering the system can be calculated with the classical formula for a fluid of constant density at a given depth.

$$P_{in} = \rho g h \approx 9.78 \quad [kPa] \tag{6.11}$$

where $\rho \approx 997\,[kg/m^3]$ represents the density of the fluid -which was slightly underestimated using the clear water density-, $g \approx 9.81\,[m/s^2]$ is the Earth's gravity, $h \approx 1.0\,[m]$ corresponds

to the height of the feed tank from the ground -approximately 1 meter-.

Permeate Pressure The pressure on the permeate side is open to the atmosphere, so their difference should be small. As expected, the P_p experimentally was always measured in the range $[18, 30]\,[kPa]$, and, for the sake of simplicity, modeled as a constant (taking its median). In reality, the permeate pressure is not constant rather it has up and down jumps which seemed unpredictable or at least uncorrelated to any variable. In practical terms, it's not a big issue because the possible range of values is small and its values are small as well ($\leq 0.3[bar]$) causing in the worst-case scenario an error $< 0.1[bar]$ which is negligible.

$$P_p \approx 21.88 \;\; [kPa] \tag{6.12}$$

6.4 Discrete variables

Discrete variables are described with equations, and no circular dependencies are present. In the literature was not found any general model for calculating the behavior of the pressures. Still, plotting them on a chart makes it clear -in our setting- that knowing a pressure allows us to approximate the others via a linear transformation. So, all the pressures are obtained by adding an offset and multiplying by a scalar the retentate pressure (P_r). In our experiments, we increased (or decreased) the retentate pressure by closing (or opening) the valve on the pipe that loops back the retentate to the feed tank. Moreover, the retentate flow as well as been assumed to be discrete, since having an almost constant value at a given TMP rate.

The data can be summarized in four macro-areas: (1) clear water (with concentration $\approx 0.0\,[g\,/\,L]$), (2) with feed concentration $\approx 0.49\,[g\,/\,L]$, (3) with feed concentration $\approx 1.30\,[g\,/\,L]$, (4) with feed concentration $\approx 1.71\,[g\,/\,L]$. Moreover, inside each macro area, many different TMPs have been tested. Our biggest mistake was *impatience*: we should have waited more time (at least 15-20 mins) before changing the pressure. That's because we would have been able to understand when and how the flux reaches its fix point. Instead, sometimes we changed it after 2-3-4 minutes which is clearly not enough. Moreover, we would also have more certainty and precision on estimating the discrete variables.

Retentate Pressure The retentate pressure is the input parameter that the operator can arbitrarily choose to increase or decrease, altering the entire process. So, there is no need to define any sort of function to calculate it. Choosing how and when the retentate pressure should be updated is the core of our work and the goal of our controller synthesis problem. In picture 6.1, is visible the chart that shows that indeed the retentate pressure can be modeled as a discrete variable. After each update of the retentate pressure, the median is re-computed. Thus, what emerges is a set of constant-pressure executions (arbitrarily short or long). We report the evaluation metrics that use the median of the retentate pressure in each constant-pressure interval as an estimator for the variable itself. `R-Squared` $= 0.9995$, `MAE` $= 1.56\,[kPa]$, `MAPE` $= 1.16\%$, `maxAE` $= 15.19\,[kPa]$, `maxAPE` $= 8.94\%$, and `RMSE` $= 2.39\,[kPa]$.

First Feed Pressure Is the pressure applied by the pressure pump, before entering the circulation pump. The pressure is approximated via a linear model of the retentate pressure. The

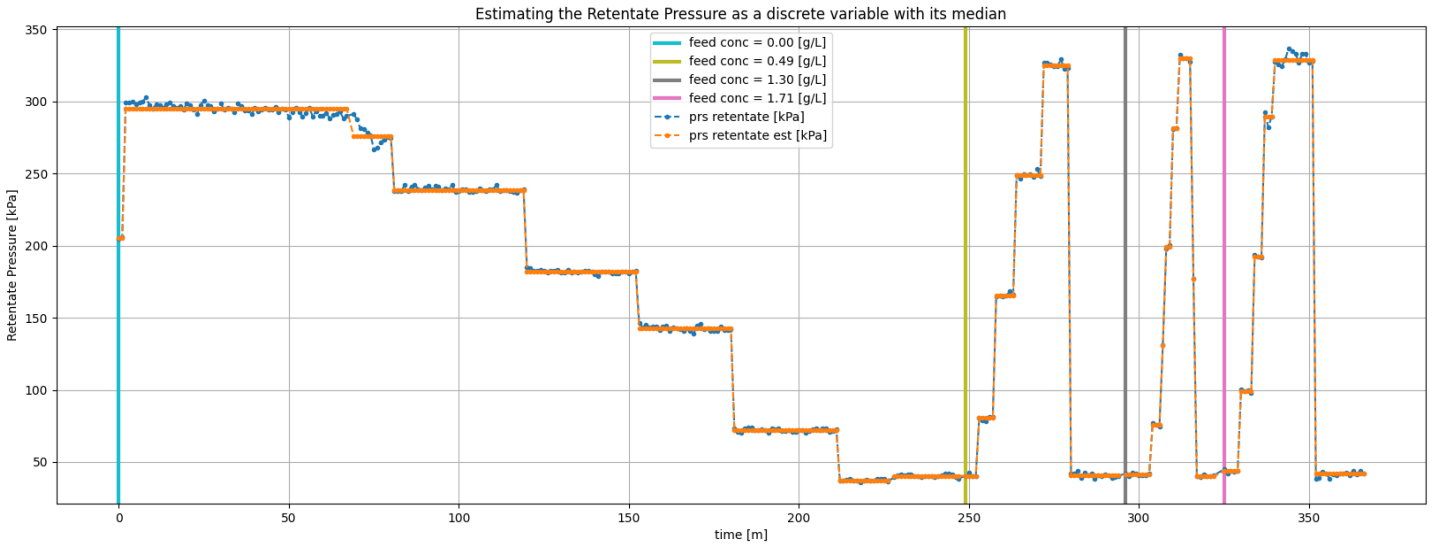


Figure 6.1: Retentate pressure approximated to a discrete variable using the median value of each unchanged interval. Maximum error of 8.94% (15.19 $[kPa]$).

model is simple and accurate, it includes all the data (both for clear water and wastewater). All the metrics are excellent: `p-values` $<$ 1e$-$300 for both the intercept and the pressure, `Adj R-Squared` $= 0.999$, `MAE` $= 1.66\,[kPa]$, and `MAPE` $= 0.94\%$, `RMSE` $= 2.32\,[kPa]$. Since `maxAE` $= 14.79\,[kPa]$ and `maxAPE` $= 4.61\%$, the model achieves a great level of precision on each sample (the maximum error is always $< 0.15\,[bar]$. Picture 6.3 enables visually confirming what summary statistics say.

$$P_{f_1} = 34.08 + 0.986\,P_r \;\; [kPa] \tag{6.13}$$

Second Feed Pressure It is the pressure the circulation pump applies, i.e., the pressure at which the feed water reaches the membrane. The same considerations said for the precedent model hold also in this case. All the metrics are fine: `p-values` $= 0.0$ for both the intercept and the pressure, `Adj R-Squared` $= 0.998$, `MAE` $= 3.29\,[kPa]$, and `MAPE` $= 0.8\%$, `RMSE` $= 4.11\,[kPa]$, `maxAE` $= 13.53\,[kPa]$ and `maxAPE` $= 2.91\%$. Picture 6.3 shows this model.

$$P_{f_2} = 260.16 + 0.996\,P_r \;\; [kPa] \tag{6.14}$$

Retentate Flow The retentate flow is the rate at which the retentate goes back into the feed tank. This formula is indeed an approximation of reality and with a bigger dataset with more samples and the concentration attached, we believe it could be greatly improved. First, it has no mechanical meaning: the flow is expressed in $[L/h]$ while the pressure in $[kPa]$. Second, different from the two pressures above, to improve the linear model's precision, the sample data of clear water were not included. That's because clear water does not cause fouling thus the permeate flow will be much greater thus reducing (and bad-predicting) the retentate flow.

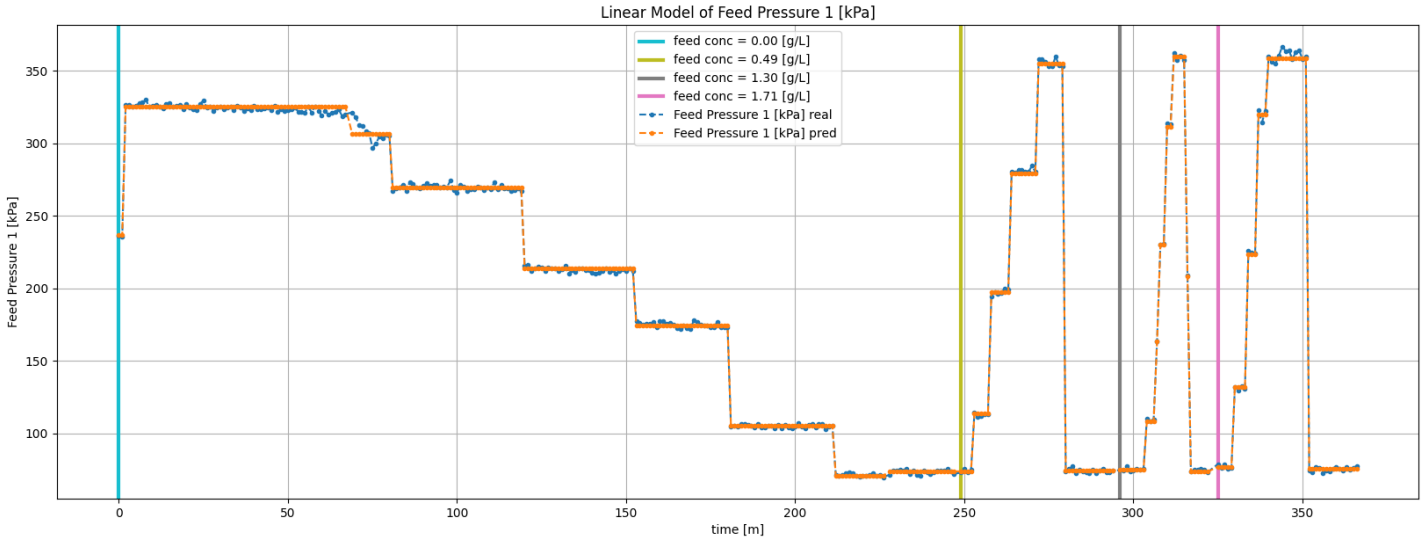


Figure 6.2: Linear model that approximates the feed pressure pumped by the pressure pump based on the retentate pressure with a maximum error of 4.61% ($14.79\,[kPa]$).

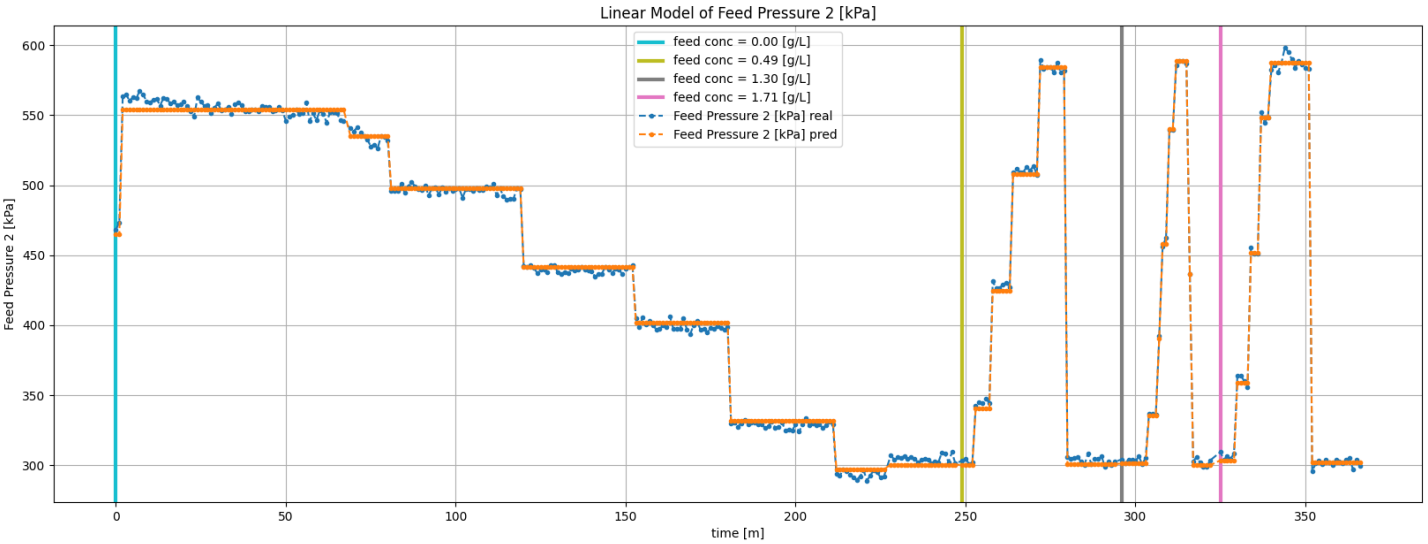


Figure 6.3: Linear model that approximates the feed pressure on the membrane side (pumped by the circulation pump) based on the retentate pressure with a maximum error of 2.91% ($13.53\,[kPa]$).

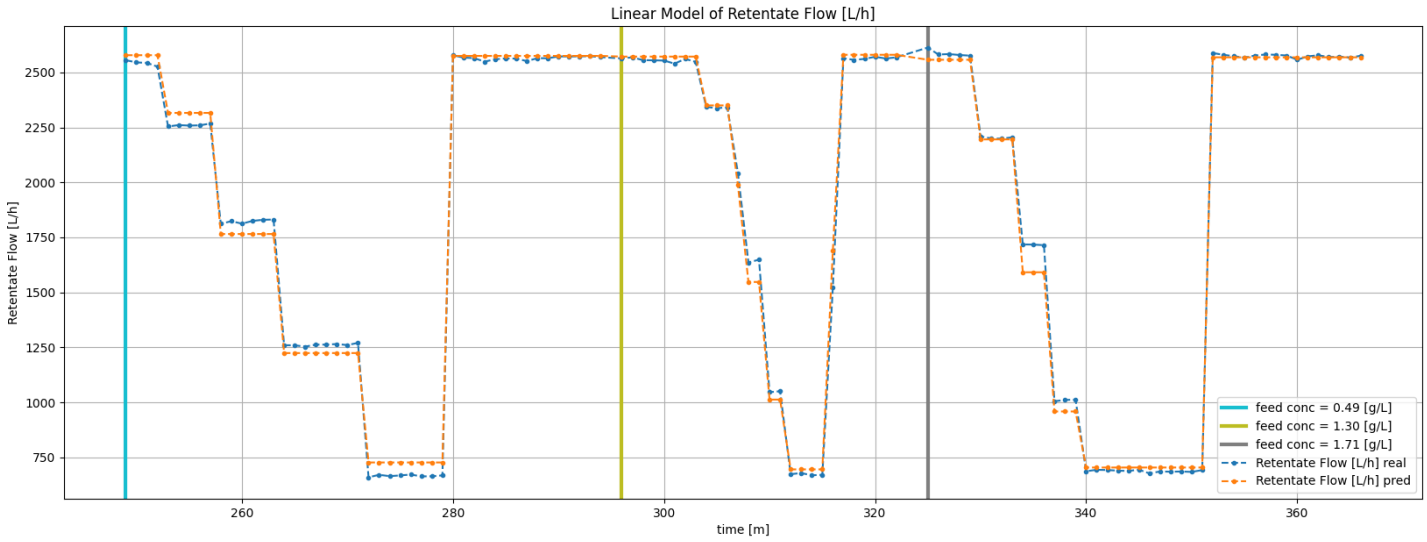


Figure 6.4: Linear model that approximates the retentate flow based on the retentate pressure with a maximum error of 10.95% (166.91 $[L\,/\,h]$). The samples belong to macro-areas 2,3,4 while the first one (not fouled, with clear water) was removed.

Yet, with the data currently available no better models were found. We also tried to include many factors even with a non-linear transformation (such as `log`, `exp`, `power`, `root`, etc.) but none with enough statistical significance. This model, excluding clear water samples, has great `p-values` $<$ 1e$-$100 for both the intercept and the pressure, `Adj R-Squared` $= 0.997$, `MAE` $= 30.72\,[L\,/\,h]$, and `MAPE` $= 2.36\%$ but `RMSE` $= 42.46\,[L\,/\,h]$, `maxAE` $= 166.91\,[L\,/\,h]$ and `maxAPE` $= 10.95\%$ are not so much satisfactory . In particular, picture 6.4 highlights that the model underestimates the flow when it lies in the interval $[1000, 2000][L\,/\,h]$, that is the less represented interval (thus, the model is quite biased and favors low or high values).

$$Q_r = 2841.02 - 6.50\,P_r \;\; [L\,/\,h] \tag{6.15}$$

Transmembrane Pressure The transmembrane pressure can be calculated directly given the other pressures as shown in equation 6.1, amongst the axioms. Thus, no linear model is needed.

6.5 Continuous variables

They are described with differential equations.

Temperature In the absence of a cooling system -as in our case-, the temperature will increase over time (due to the heating of the pumps). Experimentally, it is an almost perfect linear trend over time. In the first two minutes were observed jumps in the temperature, but due to the lack of enough data to guarantee statistical reliability, the phenomenon was simply ignored. Of course, it's a big simplification but, to understand this behavior, more samples are required. So, in the model, only the constant slope was kept from the first 2 minutes onwards, which is quite stable. The coefficient 1.189e$-$1 is the average of the estimated coefficients in

the 5 experiments. Each of them have a coefficient between 0.10 and 0.13 (so, quite similar), `p-values` $<$ 1e$-$18, `R-squared` $>$ 0.97, `MAPE` $<$ 1.5%. The initial temperature instead depends on the environmental condition and makes no sense to fix it *a priori*. Moreover, consider that, in real-world scenarios, the temperature is a controlled factor (fixed), and it would have a slope 0.0.

$$temp = temp^{(0)} + 1.189\text{e}{-1} * t \;\; [^\circ C]$$

From which we can derive its derivative:

$$temp' = 1.189\text{e}{-1} \tag{6.16}$$

which means that there is an increase of almost $1.2\,^\circ C$ every 10 minutes.

Permeate Viscosity The viscosity depends on the pressure (constant for the permeate), the temperature (continuous), and the concentration. Since in our experiments, the concentration of the permeate was always < 0.5 $[g/L]$ we assumed, for the sake of simplicity, to consider the viscosity of the pure water, ignoring the concentration. In case of an increasing temperature, the viscosity will decrease, making it necessary to take into account its derivative μ_p'. The package `CoolProp`, available in Python, contains a function `PropsSI()`, that calculates many thermophysical fluid data, viscosity included. Thanks to it, the viscosity has been calculated for any temperature in the range $[5, 70]$ $[^\circ C]$ at intervals of 0.25 $[^\circ C]$, at the pressure of $P_p + 101.325$ $[kPa]$. To transform the above discrete series into a continuous one, some linear models were tried. Finally, for the specified interval, an almost perfect model was detected using the temperature and its squared root as factors. The evaluation metrics show that the model is almost perfect: `R-squared` $= 0.9997$, `p-values` $<$ 1e$-$200 for both intercept and coefficients, `MAE` $<$ 4e$-$6 $[Pa\,s]$, `MAPE` $= 0.44\%$, `maxAPE` $= 1.81\%$. As visible in picture 6.5, there is a slight overapproximation at low temperatures (< 8 $[^\circ C]$) (with no real consequence since in our data the temperature is always above).

$$\mu_p = 2.2436\text{e}{-3} + 1.4968\text{e}{-5} * temp - 3.4561\text{e}{-4} * \sqrt{temp} \;\; [Pa\ s]$$

From the linear model we can derive the water viscosity derivative:

$$\mu_p' = 1.4968\text{e}{-5} * temp' - 3.4561\text{e}{-4} * \frac{temp'}{2\sqrt{temp}} \tag{6.17}$$

Total Resistance (under constant pressure) The equation is obtained by deriving equation 4.7, after bringing R_{tot} to the LHS and J to the RHS. The TMP is treated as a constant while the viscosity and the flux are continuous variables. So, under constant-flux operations, the equation should account for a non-constant TMP.

$$R_{tot}' = \frac{d}{dt}\frac{TMP}{\mu_p\, J} = -\frac{TMP}{\mu_p^2\, J^2} * \frac{d(\mu_p\, J)}{dt} = -\frac{TMP}{\mu_p^2\, J^2}\left(\mu_p\, J' + \mu_p'\, J\right) = -\frac{TMP}{\mu_p\, J}\left(\frac{J'}{J} + \frac{\mu_p'}{\mu_p}\right) \tag{6.18}$$

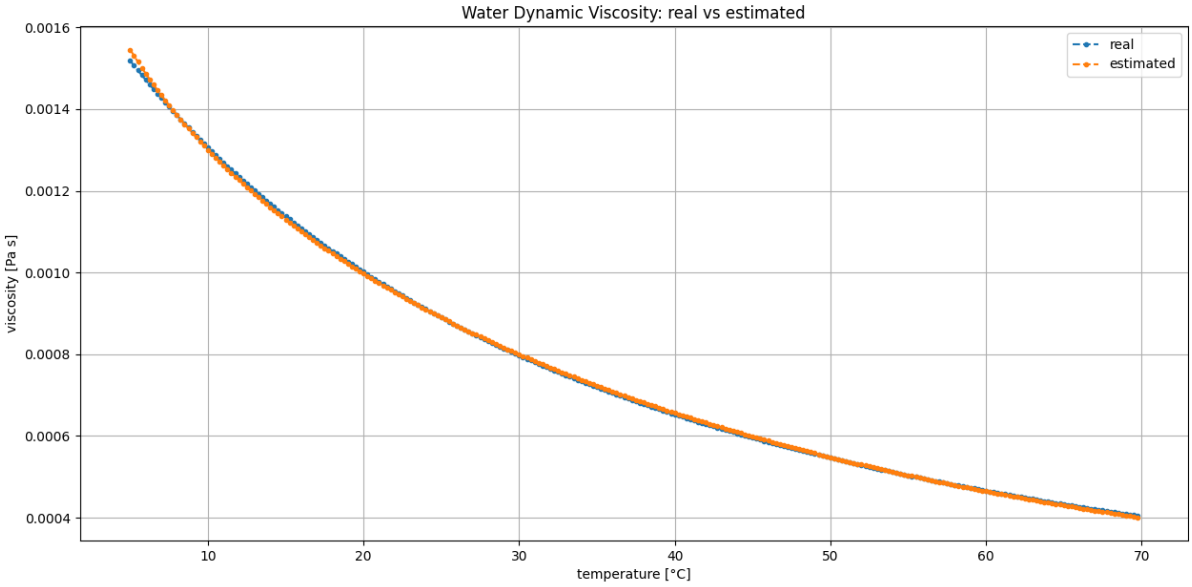


Figure 6.5: Linear model that approximates the pure water viscosity at the constant pressure of 101.347 $[kPa]$ in a temperature range of $[5, 70]$ $[°C]$.

Permeate Flow The equation can be trivially deduced given 6.2, which shows how the two variables differ by the membrane area (a constant scalar).

$$Q'_p = J' \tag{6.19}$$

Feed Flow A direct consequence of 6.4: applying the derivative on both sides, the retentate flow -being a discrete variable- will have a null derivative ($Q'_r = 0$) and so only Q'_p will remain.

$$Q'_f = Q'_p = J' \tag{6.20}$$

Feed Pump Energy The energy consumed per hour by the feed pump can be calculated knowing the power of the pump: $W_{fp} = Q_f * P_{f_1}$ and remembering that the energy is the integral of the power. So, the derivative of the energy is the value of the power itself.

$$E'_{fp} = \frac{d}{dt} \int Q_f * P_{f_1} \, dt = Q_f * P_{f_1} \tag{6.21}$$

Circulation Pump Energy The same reasoning applies to the circulation pump as well.

$$E'_{cp} = \frac{d}{dt} \int Q_r * (P_{f_2} - P_{f_1}) \, dt = Q_r * (P_{f_2} - P_{f_1}) \tag{6.22}$$

6.6 Cost Function

The cost function f is defined as the minimization problem of the specific energy consumption metric. As presented in equation 6.10, the SEC is the ratio between the energy required and the permeate

volume obtained.

$$f = \min SEC = \min \frac{E_{tot}}{V_p} \tag{6.23}$$

7

Tools and Technologies

7.1 Uppaal

Uppaal is a tool for modeling, simulation, and verification of real-time systems, jointly developed by Uppsala University and Aalborg University. Since its first release in 1995, Uppaal has been successfully employed in many real-world problems. Uppaal has been refined and improved over the years, and it is currently at version 5.0 as of 2024.

7.1.1 Uppaal

Uppaal has been initially implemented for systems that could be modeled "*as a collection of nondeterministic processes with finite control structure and real-valued clocks, communicating through channels and (or) shared variables*"[39]. In other words, from the beginning, Uppaal supports systems modeled with networks of Timed Automata or Linear Hybrid Automata (which can be reduced to TA). Once the system is built, the tool allows verifying properties on it, thus, it is one of the first effective model-checkers implemented.

Uppaal consists of three main parts: a description language, a simulator, and a model-checker. The **description language** is used to define the system model and its properties. A GUI is available to construct the timed automaton. Many useful features have been included to ease the modeling phase of the automata such as synchronization and broadcast channels, urgent and committed locations, etc [12]. The **simulator** enables an interactive visualization of possible traces which results quite useful for debugging -in the earlier stages- and for understanding how and why counterexamples are reached if the checker has found any faults. Nowadays, are present two different simulators: the concrete and the abstract. The **model-checker** performs the exhaustive analysis of any behaviour. It checks if the goal nodes are reachable from an initial configuration, respecting the constraints on clocks' evolution, invariants, and guards. Nowadays, Uppaal is formed by two components in a *client-server* architecture: the GUI which is the client that communicates with the model checker, and the server that can be either local or remote.

7.1.2 Uppaal TiGa (Timed Games)

In 2005, one notable extension to Uppaal was TiGa. Uppaal-TiGa is the first efficient tool supporting the analysis of timed games allowing the synthesis of controllers for control problems modeled as timed game automata and with safety control objectives [17],[10]. Systems are modeled as a network of Timed Game Automata, where edges can be marked either as controllable or uncontrollable. Basically, the uncontrollable edges represent the external environment while, for the controllable ones, the controller can decide if to take such transitions. The synthesizer then looks for a winning strategy given a specification formula (such as reaching a given state of the automaton).

7.1.3 Uppaal SMC (Statistical Model Checking)

In 2012, another milestone was added to the tool: Uppaal-SMC, a realization of Statistical Model Checking techniques [24]. As already explained in 2.4, with SMC is possible to estimate the expected time and costs, and to look for bounded reachability probabilities, with arbitrary confidence. Supported systems could be modeled using a network of either Priced Timed Automata (PTAs) [16], Stochastic Timed Automata (STAs), Hybrid Automata (HAs), or Stochastic Hybrid Automata (SHAs) [22].

With regard to Hybrid Automata, Uppaal-SMC has two great advantages: (i) even if model checking on the constructed automaton is undecidable, it can still be checked with SMC; (ii) is the first tool that supports Stochastic Hybrid Automata (SHA) with ODEs as flow conditions. To enable point (ii), the ODEs are allowed to depend not just on discrete variables but also on the value of other clocks. Currently, the ODEs are not yet solved exactly but are resolved by making use of a fixed-time step Euler's integration method, which is sensitive to the discretization of the step size. Of course, if clocks are defined via a general ODE, then the classical model checking is impossible and so is disabled (unless the non-linear clocks are used only for estimating costs with no possibility of controlling any transition).

Note that Uppaal-SMC is a tool that allows the representation of a system that combines concurrency (as networks of automata), non-deterministic transitions, real-time aspects (timed constraints), and probabilistic transitions. Mixing all of these four ingredients together is not trivial and may result in inconsistencies and underspecifications, as noted by Bohlender et al. in 2014 [15]. In their work they compare Uppaal and Modes [14], the two tools that allow combining such features, and have highlighted that can happen that the computed probability of particular queries on an identical automaton may have different outcomes in the two tools, due to different assumptions made by the developers. So, considering the complexity of the modeled systems, the user must be aware of the underlying strategies applied by the tool used, in order to avoid misinterpretations.

7.1.4 Uppaal Stratego

Finally, in 2015, Uppaal-Stratego [23] was published. Stratego introduces another key feature in the Uppaal framework. Uppaal TiGa and SMC allow to synthesize strategies for a given specification. However, given a strategy (manual or synthesized), Uppaal did not support the possibility of examining non-mandatory properties and neither optimizing a non-deterministic safety strategy given a cost function. Both these missings were included in Stratego which integrates Uppaal, Uppaal-SMC and Uppaal-TiGa, consenting to generate, optimize, compare and explore different synthesized strategies.

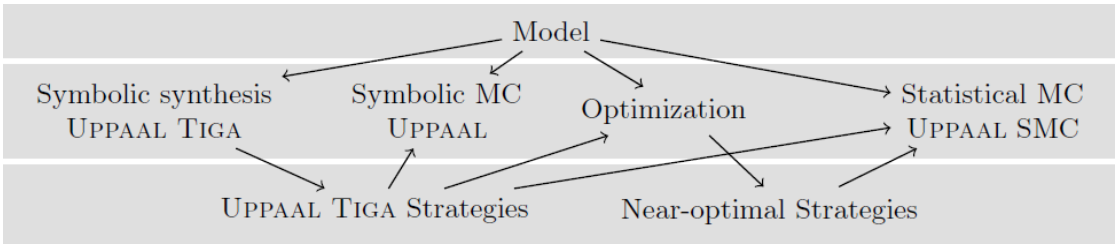


Figure 7.1: The main components of Uppaal (David et al., 2015) [23].

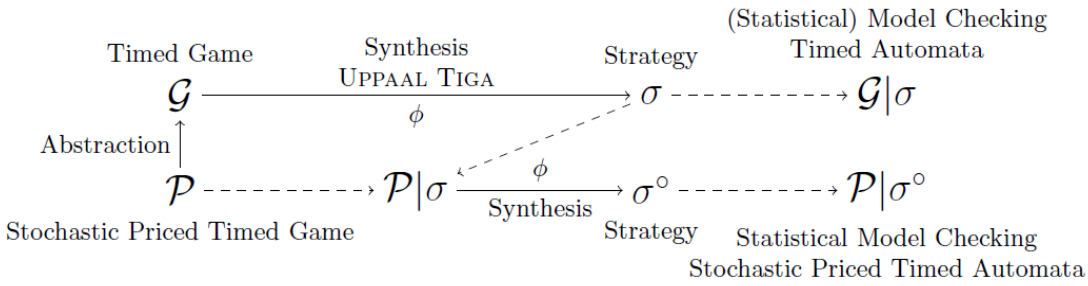


Figure 7.2: The process for synthesizing and optimizing a Stochastic Priced Timed Game (SPTG). Solid arrows represent transformations, dashed arrows reuse (David et al., 2015) [23].

Picture 7.1 shows the main components of Uppaal and how they can interact with each other.

In Stratego, a system can be modeled via a Stochastic Priced Timed Game (SPTG) $\mathcal{P}$. To synthesize an exhaustive strategy it is necessary to abstract from prices and stochasticity (which are simply ignored) to obtain a Timed Game (TGA) $\mathcal{G}$. Note that in $\mathcal{P}$ there is 1-½-players game meanwhile the abstracted version $\mathcal{G}$ is a 2-players game because the stochastic transitions are now under the control of an (artificial) opponent. Given $\mathcal{G}$, with Uppaal-TiGa it is possible to synthesize a strategy σ. Then, by applying the strategy σ on the original automaton $\mathcal{P}$ ($\mathcal{P}|\sigma$) is possible to use Uppaal-SMC to analyze the costs -initially ignored- and estimate expected values, ranges, etc. Moreover, it is possible to synthesize a statistical strategy σ^0 under σ, where, using learning methods tries to minimize the expected costs. Of course, being simulation-driven and reinforcement-learning based, σ^0 may be a sub-optimal strategy. Again, one can analyze how $\mathcal{P}|\sigma^0$ performs. Note that, when applicable, this approach permits to guarantee formally the safety requirements (with strategy σ) and to quasi-minimize the costs with strategy σ^0 under σ. Picture 7.2 visually shows the explained workflow.

Last but not least, even non-linear hybrid models can have an optimized strategy: not an exhaustive one, but again, using the learning methods a statistical strategy first is synthesized, and then it can be improved to be sub-optimal w.r.t. to the costs.

8

Implementation

The implementation can be divided into two main parts. The first one aims to analyze the real data in order to verify relationships between variables and estimate parameters. The second part is the modeling in Uppaal, plus the verification of properties and the synthesis of strategies. The souce code for the analysis in Python as well as the Uppaal model can be found in the linked GitHub repository.

8.1 Preprocessing and Data Analysis

This part regards the preprocessing, the analysis, and the parameter estimation of the real data obtained from the sensors. The code is written in Python and uses its data analysis libraries': pandas, matplotlib, statsmodel, numpy, sklearn, etc. To ensure easy reproducibility, this component is stored within a Docker container and the packages are handled with Conda. Docker provides a consistent, portable environment, isolating the application and its dependencies from the host system. Conda manages dependencies within this environment, allowing for isolated, cross-platform compatibility and simplified package installation. Together, Docker and Conda ensure that the software runs reliably and consistently across different systems. The code is divided in different python files and notebooks:

- **constants**: contains the import instructions of the used libraries plus various constants.
- **functions**: is the core of the project. All the Python functions defined and used in the notebooks are stored here.
- **1_extend**: the first step augments the raw data with new columns that can be derived, such as the viscosity, the total resistance, etc. The columns are computed using the various equations shown in section 6.
- **2_explore**: the second step is the exploratory data analysis (EDA) which consists of plotting variables, looking for patterns and dependencies, etc. From now on, some filters are applied that remove possible outliers, due to unlikely values recorded (completely out of scale or simply very far from the neighbours points). By default, the check and removal of outliers is (automatically) performed, and if any is spotted the whole data at that time instant is dropped. The user can customize the parameters that handle outliers: specify the accepted range of values of a series, or

the maximum ratio between each point and the average of its neighbours. Otherwise, the user is free to keep them.

- **3_estimate_factors**: in the third step are defined all the linear models used to estimate the discrete variables. The median retentate pressure is used as the predictor variable, and the other pressures and the retentate flow are derived from it (see section 6.4).

- **4_ARIMA_simulations**: the fourth step aims to enlarge the flux time series over larger periods. The ARIMA methods are employed, as detailed in section 5.3.

- **5_estimate_flux_min**: the fifth step focuses on finding a good linear model to estimate the flux min, as explained in section 5.4.1.

- **6_load_Uppaal_simulations**: the sixth step reads the csv file containing the simulations exported from Uppaal and converts it to a new csv easily managable. Of course, from this step on, one first has to run the Uppaal simulations. Once run, right-click on `Globals` and then `create new trajectory plots`. It will create a (useless) superimposed plot containing all the (global) variables. From there, right-click on `export` and then `csv`. The problem is that in the generated csv all the columns (each corresponding to a variable) are not aligned as one might expect. Instead, the columns are just vertically concatenated and also have with different lengths (discrete variables have fewer values). The notebook transforms the time instances from floats into integers and joins the columns together using the time as key. Then, the time instances are restored to their original value as floats. The missing discrete values are filled using their precedent value (inside the locations their value don't change). Finally, a usable csv is generated.

- **7_evaluation**: the seventh step plots the simulated data alongside the real data to verify the precision of the model.

- **main**: executes the pipeline by invoking the notebooks. The user only needs to type `python3 main.py` on the terminal. It will take several minutes and will export the output of each notebook in a separate HTML file, so one can easily check if there is something to fix. Note that each notebook contains a lot of plots and prints to ease the debug and verify the correctness of the analysis. To customize some preferences there is a JSON file where many parameters can be set according to the user's needs.

Moreover, the pipeline also exports the estimated parameters in a unique JSON file. Currently, those parameters (mainly the coefficients of the various linear models) are manually copied into the Uppaal model. This task can be easily automated since Uppaal already supports the execution of functions from other programming languages.

8.2 Uppaal Modeling

Once, the data analysis is done, and relationships and coefficients are defined we have a complete mathematical model. The remaining task is to build the Uppaal model that correctly implements the

mathematical model. Finally, once the automaton is built, one can verify properties and synthesize strategies out of it. The implemented model is made only of one automaton which has four locations:

1. **Start**: Initial location, where all variables are set to zero. Instantaneously, the transition towards the controller state is taken and the transition executes the function that initializes all the variables according to the given parameters.

2. **Controller**: represents the controller, which decides whether to modify the pressure or keep it as it is. Being an *urgent* state (see the "u" inside the state circle), it must choose a transition instantaneously (hence no time is elapsing).

3. **Filtering**: represent the plant. By remaining in the location the time goes on and the filtering process proceeds. After a fixed time interval -defined by the constant `UPDATE_INTERVAL`- (for example 5 minutes), the invariant forces to leave the state and to take a transition towards the controller state. When the feed volume reaches its minimum value in the tank, the filtering process is considered finished and the end state is immediately reached thanks to the urgent broadcast channel `ASAP`. The discussion on how and why we implemented in this way the `UPDATE_INTERVAL` rate and the lower bound on the value of the feed volume already in section 2.5.4.

4. **End**: the filtration is ended, and the system stops.

The key transition is the one from the controller state to the filtering state. There are two possible transitions: the `Keep Pressure` and the `Increase Pressure`. The `Keep Pressure` does not alter anything. The `Increase Pressure` instead increments the retentate pressure. We excluded the case of decreasing the pressure because from the theory of filtration is known that by decreasing it there is no particular gain. Anyway, one is free to change the `MIN` parameter to a negative value and see what happens by allowing such behaviour. Thanks to Uppaal syntax, instead of creating a set of transitions each with a specific value for the increase, its value can be chosen from a set of possible candidates using the `select` operator. Note that this operator is just syntactic sugar built over the basic syntax of a Hybrid Automaton, but the expressive power is still the same. Given a set of possible values, the chosen one is stored in the `increase` variable. The `increase` variable represents the amount of pressure that will be added to the retentate pressure (or eventually removed) and emulates the behaviour of the pressure valve positioned along the retentate stream. After this, the function `update_variables` is executed which takes in input the current `increase` value and updates the TMP and all the other variables are updated accordingly.

We defined the set of candidate values to be selected using three constants: a minimum value (`MIN`), a maximum value (`MAX`), and a multiplier (`MUL`) that scales all the integers inside the min to max interval. After setting `MIN` $= 1$, `MAX` $= 5$, `MUL` $= 25$, the resulting interval for the variable `increase`$\in [MIN = 1, MAX = 5] * MUL = 25\ [kPa] = \{25, 50, 75, 100, 125\}\ [kPa]$. Note that these parameters can be easily customized by the user, even including negative values to allow the decrement of the pressure.

Increasing the number of possible choices can help find the best sequence of actions to take, but, on the other side, increases exponentially the complexity. Every time we reach the control state there are $k + 1$ possible edges to take, where 1 represents the non-updating transition and k is the number of possible choices in the case of an update. So, the set of all traces can be represented with a tree with size

$(k+1)^i$, where i is the number of times the control state is reached. Thus, there is a trade-off between time and cost efficiency in choosing the set of possible choices: increasing the set of choices may help find cheaper strategies, but the time needed to find those strategies will be exponentially increasing. In truth, if we limit the choices to be only positive (thus excluding the case of decrementing the pressure) $(k+1)^i$ would be an overapproximation because there is a fixed upper bound for the TMP, which corresponds to the guard `TMP < TMP_MAX - 1`. So, as time goes on, the available choices could decrease if the TMP keeps growing, up to a point where the only remaining transition is the `Keep Pressure` transition.

8.2.1 Why Uppaal-Stratego

In Uppaal one can (a) manually run the simulations, setting at each step the desired pressure, (b) execute random simulations, (c) verify the statistical properties of the simulations (using stochastic temporal logic), and (d) synthesize sub-optimal strategies.

The **concrete simulator** implements the simulations and has a user-friendly GUI composed of 5 main blocks: (1) the automaton with highlighted the current location, (2) the list of enabled transitions, (3) the list of all the variables with their current value, (4) the current trace, and (5) the plot of the evolution of one variable over time. With it, one can proceed either manually one step at a time - choosing a transition in case of non-determinism- or randomically and see how the automaton evolves over time, track how each variable changes, go back to a previous location, etc.

The great advantage of this tool is not the possibility of simulating the process, but rather synthesizing strategies with **Uppaal-Stratego**, which relies on statistical model checking (SMC). Given an objective function f, Stratego launches thousands of simulations and uses Reinforcement Learning to learn how to minimize (or maximize) f (being based on machine learning, the strategy could be sub-optimal). Once a strategy s is found, one can set the current strategy to be s (by default is none) and then, under the restrictions imposed by s, perform analysis with the concrete simulator, verify new properties, or even look for a new strategy s' under s.

Actually, our model is not stochastic at all, thus talking about probabilities and statistical properties may sound misleading. The problem is that the hybrid automaton here described, cannot be symbolically verified in Uppaal. Uppaal recognizes that cannot solve the automaton and disables all the available features of classical symbolic model checking.

So, we need to rely on a non-symbolic method, i.e., statistical model checking. To do so, it is necessary to deal with non-deterministic transitions because traditionally SMC is applied when the transitions are stochastic. In our automaton, non-determinism arises in the selection of the $k+1$ possible choices from the controller state to the filtering state. The default approach implemented in Uppaal is to assign a uniform probability to the enabled edges (actually, it's quite more complex, for details see [16], [22]). Thus, the probabilities are the result of a stochastic interpretation of non-determinism. Note that, in Uppaal-Stratego, unlike the concrete simulator, only the non-deterministic non-controllable transitions are viewed as stochastic transitions with uniform probability. The controllable edges represent instead the actions that the learning agent can control to find the strategy.

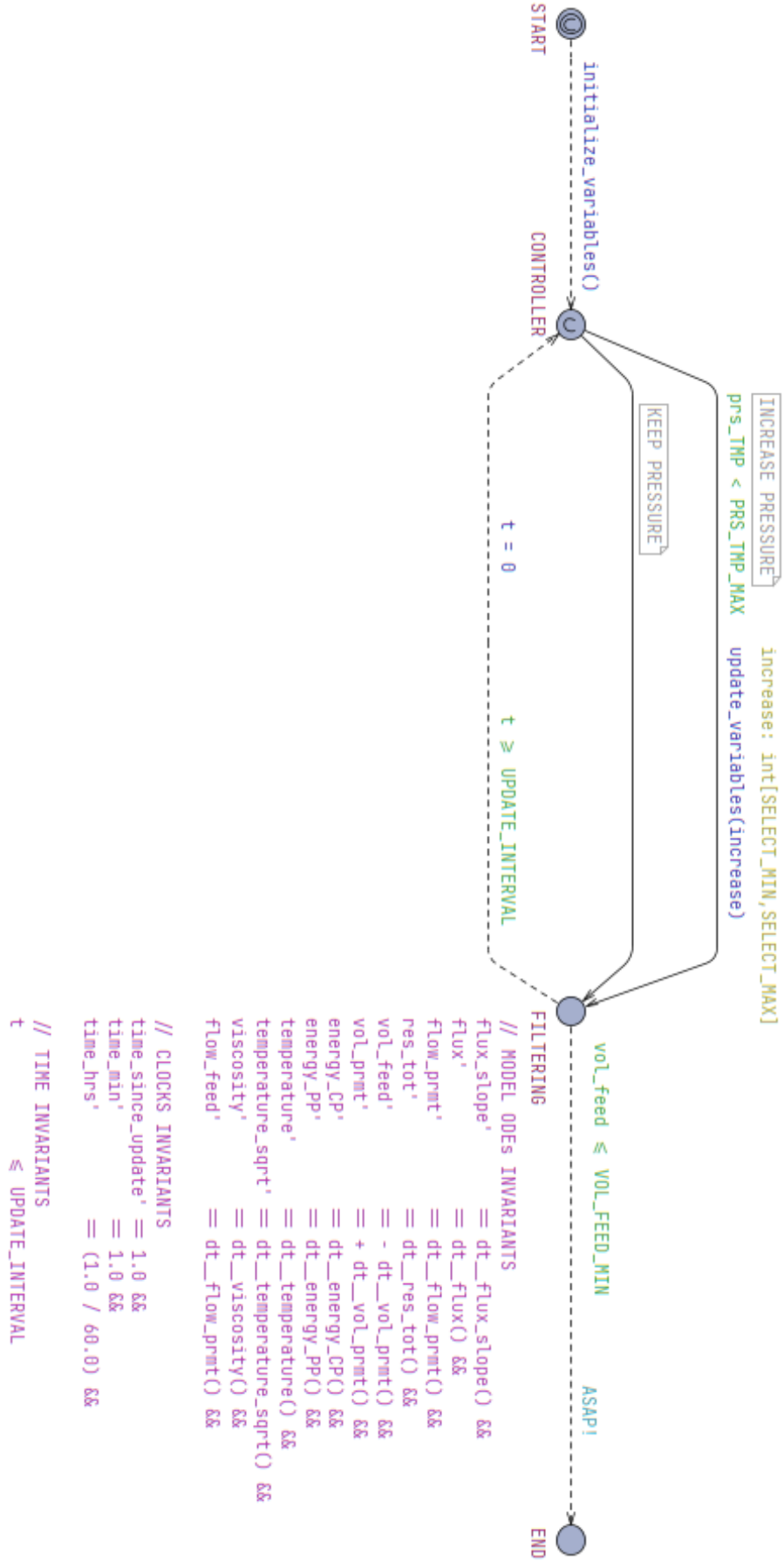


Figure 8.1: The Uppaal implemented model, solid arrows are controllable transitions, and dashed arrows are non-controllable. Guards are in green, assignments are in blue, invariants are in purple, and channels in light blue.

8.3 Uppaal Strategies

The Uppaal's syntax to define SMC strategies is the following:

$$\text{strategy } name = \text{Goal [UpperBound] Observability : TemporalExpr StrategyConstraint}$$

$$where$$

$$\begin{aligned}
\text{Goal} &::= \{ \text{ minE} \mid \text{maxE} \} \ (expression) \mid \text{minPr} \mid \text{maxPr} \\
\text{UpperBound} &::= \{ \mid clock \mid \# \} \leq value \\
\text{Observability} &::= \mid '\{' \ discrete_variables \ '\}' \ \rightarrow \ '\{' \ continous_variables \ '\}' \\
\text{TemporalExpr} &::= \{ \ \Box \mid \Diamond \ \} \ expression \\
\text{StrategyConstraint} &::= \mid \text{under } name
\end{aligned}$$

- **Goal**: represents what should be minimized or maximized. If the goal is to min/max a function then the syntax is `minE (maxE)` *expression*, where the expression is our goal function. If instead the objective is to min/max the probability of the TemporalExpr then should be written only `minPr (maxPr)`.
- **UpperBound**: defines when the sample traces should be stopped. It can be defined as the maximum value of the global time ($\leq t$), of a specific clock ($clock \leq t$), or with the maximum number of transitions ($\# \leq n$). Note that ($\# \leq n$) guarantees termination, ($\leq t$) guarantees termination if time diverges (automaton is non-Zeno), ($clock \leq t$) does not guarantee termination, and must be assured by the modeler.
- **Observability**: defines the set of variables that Stratego should consider in order to minimize or maximize the goal. If it is empty implies full observability, which finds the optimal strategy for the given set of samples but can be quite slow. By defining a subset of the variables to watch the process can gain a significant speed up.
- **TemporalExpr**: defines the temporal expression that should be satisfied while looking for the goal function. The syntax of the TemporalExpr is similar to a restricted version of MITL formulas: only one temporal operator can (and must) be specified at the beginning, and then only an expression is allowed, made of constants, variables, locations, etc.
- **StrategyConstraint**: defines which strategy must be followed while trying to satisfy the current one. Basically, it allows combining strategies: the classical example is first to define a `Safe` strategy that avoids ending up in bad states and then to define a strategy `Optimal [...] under Safe` that optimizes the goal while respecting the safety conditions.

Once a strategy s has been synthesized, it can exported in a JSON file with the command `saveStrategy`(s). To load a previously learned strategy there is the command `loadStrategy`(s). Note that, strategies are bound to the particular instance of the implemented model and can be loaded only if the model is unchanged. If one changes anything in the model after the strategy is learned, Uppaal will throw an

error and will not load the strategy into a different model. The motivation is quite easy: if something changes in the model, you can't know if the learned strategy is still a valid solution.

9

Experimental Results

Given the small amount of samples, the model is far from being perfect. Nevertheless, being a preliminary work, it laid the foundations for future refinement when more real samples will be available. It is important to note that everything that has been implemented is a mixture of logical considerations, well-known filtration methods, and statistical significance. Any possible issue in the model has been explained, and, possibly detailed with better solutions. There already exist many *state-of-the-art* models that try to simulate the flux over time ([41] reviews and compares many). The advantage that our type of solution can offer, is that rather than just simulating the flux, it can tell a human operator how to adapt the flux depending on a goal function that can be designed according to the user's needs.

9.1 Evaluation of the Uppaal simulations

Picture 9.1 presents a plot of the real data of the flux against the flux simulated with Uppaal. To do so, we enlarged the possible updates of the pressure and reduced the interval rate at which the control can alter the pressure. We allowed the retentate pressure to be updated from a decrease of $-60 * 5 = -300\,[kPa]$ to an increase of $+20 * 5 = +100\,[kPA]$. We reduced the interval rate to every minute (instead of every 5 minutes). We also disabled the derivative of the temperature, and compared the fluxes at a fixed temperature of $20\,[°C]$ (and so at a fixed viscosity). Plus, we forced the automaton to take the right transition at the right time to emulate the same pressure changes visible in the data. Finally, we exported all the variables to a `csv` file, which we imported into Python and plotted together with the real data.

Moreover, we calculated the regression error metrics between the real flux and the one resulting from the Uppaal's simulation. Note that, for the real data, there is only one data point available per minute; thus, we filtered the Uppaal simulations, keeping only the values recorded when the time is an integer. Considering the complexity of the process, the resulting metrics are satisfactory: `R-squared` $= 0.9936$, `MAE` $= 2.94\,[LMH]$, `MAPE` $= 3.08\%$, `maxAPE` $= 10.72\%$. An average error rate of 3% and $< 11\%$ in the worst-case scenario is a promising result.

One might point out that at $t = 0$ the initial total resistance $R_{tot}^{(0)}$ is set higher than the actual value, leading to an underestimation of the first 2-3 data points. We set the $R_{tot}^{(0)} =$ 4e12 while the first recorded value is 3.3e12 representing a 20% increase. We chose to start with a higher initial value because the

resistance increases rapidly during the first few minutes as the previously clean membranes begin to foul. Accurately modeling this initial rapid fouling would require significantly more data samples, but we only have this single set since we never cleaned the membranes and repeated the experiments. Therefore, it is impossible to model this initial behavior accurately. Given that it affects at most the first 5 minutes, we decided to simply start with the resistance already increased and underestimating the first data points.

To enhance the current model, the first step is to test it with more data, possibly generated using various TMP values, concentration levels, and different types of fluids, not just ink-water dispersions. Another potential way to generalize the model is by modifying components of the plant. For instance, what would be the implications of using different membranes, pumps, pipes, or any other equipment?

More importantly, the above metrics are essentially good, especially an `R-squared` > 0.99. Nevertheless, is important to consider that, being our dataset very small, we used the whole dataset as both the training set (to build the model) and test set (to evaluate the model). Although this is not a machine learning project and there are only a few learned parameters derived from linear models, the level of generality of those is unknown.

According to eq. 4.10 , the factors that define how the flux will evolve are the transmembrane pressure TMP, the minimum flux J_{min}, the total resistance R_{tot}, the fouling method n, and its constant k_n. For the TMP, its estimation is straightforward using equation 4.7, and the pressures used to compute it are correctly estimated using a linear transformation of the retentate pressure. The model for J_{min} seems correct but the high correlation between factors must be remembered. Regarding the total resistance, its derivative is derived from the resistance-in-series model. Yet, it depends on the flux, so, if the flux is bad estimated so will the resistance. Moreover, the flux also depends on the R_{tot}, forming a circular dependency, thus, even a relatively small error in the beginning can become very large as time goes on. For the latter two, the fouling method and its constant, we haven't been able yet to identify which fouling type is occurring. Yet, we think (but we are not fully sure) it is a case of cake filtration, which implies $n = 0$. Then, assuming $n = 0$, we launched the Uppaal simulations trying different values until we found a likely candidate, which makes the whole prediction resemble the real data. In our case, we set $k_n = 5.5e - 6$.

9.2 Example of queries and strategies

As already mentioned, the most valuable benefit of this solution is the possibility to ask questions and verify hypotheses on the model. Here we'll give some real examples of MITL queries that can be verified about the plant.

The first query is quite easy and computes the probability that in a maximum of 60 time units (minutes in our case) the filtration process ends. $\mathcal{M}.l$ is the Uppaal's syntax to indicate the location l on automaton $\mathcal{M}$, while `<>` and `[]` stand for $\Diamond$ and $\Box$ operators. Thus, `<> machine.END` will be satisfied as soon as the process finishes. The `[0,60]` part is mandatory because an upper bound in terms of time or number of transitions must be always given.

```
Pr(<>[0,60] machine.END)
```

The following strategy seeks the fastest way to end the filtration process, by minimizing the total

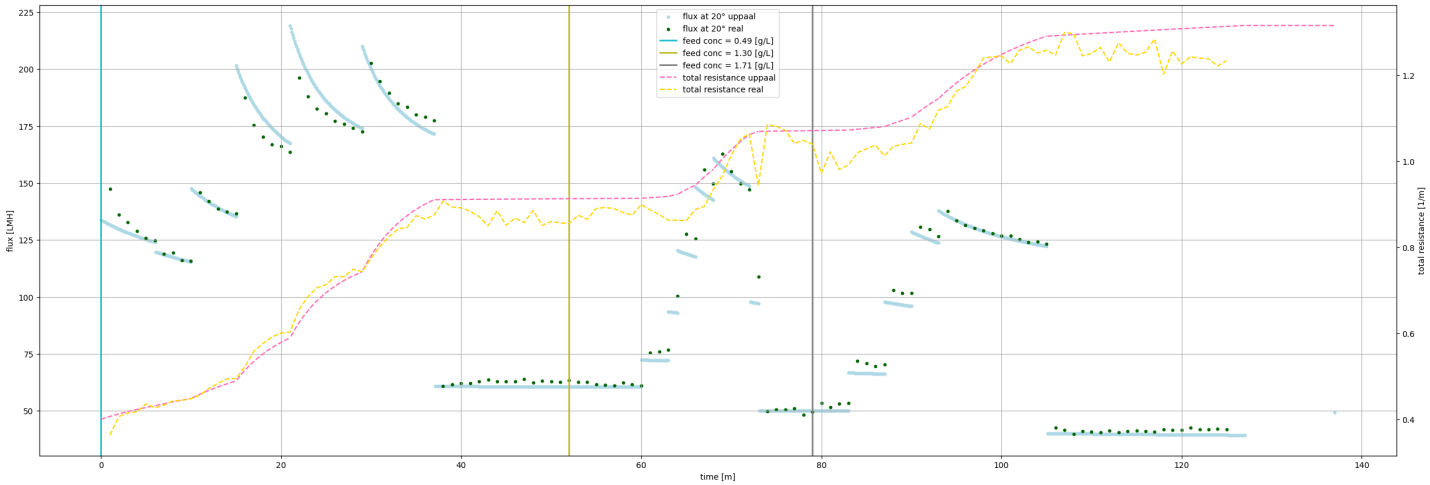


Figure 9.1: Here is shown a comparison between the flux of the real data (dots in dark green) and the estimated one in Uppaal (in light blue). Similarly, the dashed lines represent the total resistance, with the real data in yellow and the Uppaal's estimation in pink. The three vertical lines mark the time instances in which we increased the concentration of the feed bulk.

time. As one would expect, the strategy suggests to to maximize as soon as possible the TMP, so the flux is maximized as well and the filtration process cannot be faster. The MITL formula `<> machine.END` guarantees that we reach the end state, i.e., all the feed water has been filtered (excluded the minimum amount that must be always present in the tank).

```
strategy Fast = minE (time_min) [<=60] {TMP} -> {flux} : <> machine.END
```

The next strategy aims to minimize the specific energy consumption (SEC), i.e., the holy grail of this research. Actually, we do not divide by the permeate volume because the total initial volume is a fixed parameter thus we would just divide by a constant. Between the curly brackets, we added the variables that Stratego should observe during the learning process; the discrete ones must be listed on on the left while the continuous on the right.

```
strategy Cheap = minE (energy_PP + energy_CP) [<=60] {TMP} -> {flux} : <> machine.END
```

Unfortunately, also the outcome of this strategy is exactly the same as the `Fast` one: just push the TMP to the max. Unless the formulas for the energies are wrong -we do not have any real data to compare the formula with-, this result is quite unexpected. After some analysis, asking ourselves why seems convenient to always push the TMP, we obtained the following chain of implications: (1) by closing the retentate valve, (2) the retentate pressure increases P_r which causes (3) a decrease of $\Delta Q_r < 0$ to the retentate flow (recall that P_r has a negative coefficient for computing Q_r in eq. 6.15); (4) although the permeate flow increases of $\Delta Q_p > 0$, the total feed flow decreases because the increment on the permeate side is smaller than the decrement on the retentate side (in symbols: $|\Delta Q_p| < |\Delta Q_r| \Rightarrow Q_r + \Delta Q_r + Q_p + \Delta Q_p = Q'_f < Q_f$). (5) Finally, since the ODEs of the energy of the two pumps (equations 6.21 and 6.22) are directly proportional to the feed flow, a decrement of Q_f causes also the energy to decrease as well. So, basically, by increasing the TMP, the total feed flow decreases and so does the energy.

Moreover, even if everything is approximately correct, one could include additional costs into the goal function and then see what happens. For example, is is known that at high TMPs, the membranes get more fouled, making necessary longer cleaning periods and increased possibility of damaging. It could then become possible that pushing the TMP wouldn't be the more convenient thing to do (which indeed is the strategy commonly put into practice in the industries).

10

Conclusion

The goal of this project is to provide a tool that finds strategies to optimize an ultrafiltration process, in particular to reduce energy costs. Considering the rising costs and the lowering availability of freshwater, industries that daily consume large amounts of it can benefit from this research, shifting towards a more sustainable **circular economy**.

The sample data has been collected from the experiments that we conducted on a pilot plant. Based on those data, we then provided a mathematical modelization of the **ultrafiltration process**, which also includes an external controller. The model is represented by a hybrid automaton and has been implemented in Uppaal. Uppaal's simulation of the real data has an **average error of the** 3.08 %, thus we've been able to reproduce quite well the water flux in a span of 2 hours, accounting for fouling and changes in the operating pressure.

Yet, the goal of this project doesn't limit to just forecasting the flux given the initial conditions, but to take advantage of those simulations to optimize the process. With a probabilistic temporal logic, it is possible to query the model and obtain a statistically significant answer. Moreover, Uppaal Stratego supports the **definition of strategies** -such as minimizing the energy while completing a task- and finds local optima.

Unfortunately, the energy consumption doesn't seem to directly depend on the operating pressure, which instead effectively affects the amount of water filtered per unit of time. Thus, we found no tradeoff in increasing the pressure. So, it doesn't matter if the goal is to minimize either the total time or the energy consumption, because the resulting strategy is always the same, i.e., greedily maximizing the pressure as soon as possible. Even though this result may not be what we expected, we are anyway satisfied. First, we have been able to simulate the flux over time, which already is a challenging task *per se*. Second, by generalizing and extending the current model, it is possible that the optimization part would not be trivial anymore, as it is now. In particular, one could edit the cost functions, by including cleaning costs (that should be higher for higher values of the pressure) or considering the price of the energy, that we assumed to be constant but it is not always the case (tipically, it is cheaper during the nights).

11

Future Work

As previously mentioned, the main limitation of this research is the small size of the dataset used to build the mathematical model. With a larger dataset, it would likely be necessary to make generalizations and corrections to the current model. For instance, it should be verified whether the estimated linear models still hold or need to be generalized.

Specifically, regarding the minimum flux model -which is very accurate but suffers from high cross-correlation values-, it would be interesting to see how it behaves with a larger dataset. Additionally, the current model is completely independent of the concentration of the bulk feed, although it is known that this factor affects fouling. By also accounting for the concentration of the feed and the membrane, the generalized model would likely be able to accurately forecast the flux of clear water.

Another missing task is the experimental validation of the synthesized strategies. Due to some problems with the plant (see section 5.2), we were not able to run it following what the strategies prescribed. Thus, we could not check if the plant behaves as expected and if the strategies indeed minimize the energy costs or required time.

Furthermore, an interesting possibility is to stop treating the resistance as a whole and instead distinguish the baseline effects due to the membrane from those due to fouling and concentration polarization. By modeling each factor of the total resistance separately, it would also be easier to model backflushing, a method that can be used to partially clean the membranes (reducing fouling), without the need to shut down the plant and disassemble the membranes.

The most important remaining task is to compare our model's predictions for flux against those of other tools, to understand how much work is needed for it to be considered a *state-of-the-art* software.

Bibliography

[1] Gul Agha and Karl Palmskog. A survey of statistical model checking. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 28(1):1–39, 2018.

[2] Victor Aguirre-Montesdeoca, Anja EM Janssen, A Van der Padt, and RM Boom. Modelling ultrafiltration performance by integrating local (critical) fluxes along the membrane length. *Journal of membrane science*, 578:111–125, 2019.

[3] Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A Henzinger, P-H Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. The algorithmic analysis of hybrid systems. *Theoretical computer science*, 138(1):3–34, 1995.

[4] Rajeev Alur and David L Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994.

[5] Rajeev Alur, Tomás Feder, and Thomas A Henzinger. The benefits of relaxing punctuality. *Journal of the ACM (JACM)*, 43(1):116–146, 1996.

[6] Eugene Asarin, Oded Maler, Amir Pnueli, and Joseph Sifakis. Controller synthesis for timed automata. *IFAC Proceedings Volumes*, 31(18):447–452, 1998.

[7] Christel Baier, Marcus Größer, Martin Leucker, Benedikt Bollig, and Frank Ciesinski. Controller synthesis for probabilistic systems. In *Exploring New Frontiers of Theoretical Informatics: IFIP 18th World Computer Congress TC1 3rd International Conference on Theoretical Computer Science (TCS2004) 22–27 August 2004 Toulouse, France*, pages 493–506. Springer, 2004.

[8] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, 2008.

[9] Richard W Baker. *Membrane technology and applications*. John Wiley & Sons, 2023.

[10] Gerd Behrmann, Agnes Cougnard, Alexandre David, Emmanuel Fleury, Kim G Larsen, and Didier Lime. Uppaal-tiga: Time for playing games! (tool paper). In *Computer Aided Verification: 19th International Conference, CAV 2007, Berlin, Germany, July 3-7, 2007. Proceedings 19*, pages 121–125. Springer, 2007.

[11] Gerd Behrmann, Agnes Cougnard, Alexandre David, Emmanuel Fleury, Kim Guldstrand Larsen, and Didier Lime. Uppaal-tiga: Timed games for everyone. In *Nordic Workshop on Programming Theory (NWPT'06)*, 2006.

[12] Gerd Behrmann, Alexandre David, and Kim G Larsen. A tutorial on uppaal. *Formal methods for the design of real-time systems*, pages 200–236, 2004.

[13] Georges Belfort and N Nagata. Fluid mechanics and cross-flow filtration: some thoughts. *Desalination*, 53(1-3):57–79, 1985.

[14] Jonathan Bogdoll, Arnd Hartmanns, and Holger Hermanns. Simulation and statistical model checking for modestly nondeterministic models. In *International GI/ITG Conference on Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance*, pages 249–252. Springer, 2012.

[15] Dimitri Bohlender, Harold Bruintjes, Sebastian Junges, Jens Katelaan, Viet Yen Nguyen, and Thomas Noll. A review of statistical model checking pitfalls on real-time stochastic models. In *Leveraging Applications of Formal Methods, Verification and Validation. Specialized Techniques and Applications: 6th International Symposium, ISoLA 2014, Imperial, Corfu, Greece, October 8-11, 2014, Proceedings, Part II 6*, pages 177–192. Springer, 2014.

[16] Peter Bulychev, Alexandre David, Kim Gulstrand Larsen, Marius Mikučionis, Danny Bøgsted Poulsen, Axel Legay, and Zheng Wang. Uppaal-smc: Statistical model checking for priced timed automata. *arXiv preprint arXiv:1207.1272*, 2012.

[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 *CONCUR 2005–Concurrency Theory: 16th International Conference, CONCUR 2005, San Francisco, CA, USA, August 23-26, 2005. Proceedings 16*, pages 66–80. Springer, 2005.

[18] Franck Cassez, Thomas A Henzinger, and Jean-François Raskin. A comparison of control problems for timed and hybrid systems. In *International Workshop on Hybrid Systems: Computation and Control*, pages 134–148. Springer, 2002.

[19] Krishnendu Chatterjee, Marcin Jurdziński, and Thomas A Henzinger. Simple stochastic parity games. In *International Workshop on Computer Science Logic*, pages 100–113. Springer, 2003.

[20] Edmund M. Clarke, Thomas A. Henzinger, and Helmut Veith. *Introduction to Model Checking*, pages 1–26. Springer International Publishing, Cham, 2018.

[21] ZF Cui, Y Jiang, and RW Field. Fundamentals of pressure-driven membrane separation processes. In *Membrane technology*, pages 1–18. Elsevier, 2010.

[22] Alexandre David, Dehui Du, Kim G Larsen, Axel Legay, Marius Mikučionis, Danny Bøgsted Poulsen, and Sean Sedwards. Statistical model checking for stochastic hybrid systems. *arXiv preprint arXiv:1208.3856*, 2012.

[23] Alexandre David, Peter Gjøl Jensen, Kim Guldstrand Larsen, Marius Mikučionis, and Jakob Haahr Taankvist. Uppaal stratego. In *Tools and Algorithms for the Construction and Analysis of Systems: 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015, Proceedings 21*, pages 206–211. Springer, 2015.

[24] Alexandre David, Kim G Larsen, Axel Legay, Marius Mikučionis, and Danny Bøgsted Poulsen. Uppaal smc tutorial. *International journal on software tools for technology transfer*, 17:397–415, 2015.

[25] Robert W Field and Graeme K Pearce. Critical, sustainable and threshold fluxes for membrane filtration with water industry applications. *Advances in colloid and interface science*, 164(1-2):38–44, 2011.

[26] Robert W Field, Dengxi Wu, John A Howell, and Bharat B Gupta. Critical flux concept for microfiltration fouling. *Journal of membrane science*, 100(3):259–272, 1995.

[27] Robert W Field and Jun Jie Wu. Permeate flux in ultrafiltration processes—understandings and misunderstandings. *Membranes*, 12(2):187, 2022.

[28] RW Field and P Aimar. Ideal limiting fluxes in ultrafiltration: comparison of various theoretical relationships. *Journal of membrane science*, 80(1):107–115, 1993.

[29] MFA Goosen, SS Sablani, H Al-Hinai, S Al-Obeidani, R Al-Belushi, and aD Jackson. Fouling of reverse osmosis and ultrafiltration membranes: a critical review. *Separation science and technology*, 39(10):2261–2297, 2005.

[30] Sumit Gulwani, Oleksandr Polozov, and Rishabh Singh. *Program Synthesis*, volume 4. 2017.

[31] Thomas A Henzinger. The theory of hybrid automata. In *Proceedings 11th Annual IEEE Symposium on Logic in Computer Science*, pages 278–292. IEEE, 1996.

[32] Thomas A Henzinger and Peter W Kopke. Discrete-time control for rectangular hybrid automata. *Theoretical Computer Science*, 221(1-2):369–392, 1999.

[33] Thomas A Henzinger, Peter W Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, pages 373–382, 1995.

[34] Jacques Hermia. Blocking filtration. application to non-newtonian fluids. In *Mathematical models and design methods in solid-liquid separation*, pages 83–89. Springer, 1985.

[35] Winston Ho and Kamalesh Sirkar. *Membrane handbook*. Springer Science & Business Media, 2012.

[36] Alon Y Kirschner, Yu-Heng Cheng, Donald R Paul, Robert W Field, and Benny D Freeman. Fouling mechanisms in constant flux crossflow ultrafiltration. *Journal of membrane science*, 574:65–75, 2019.

[37] Myong K Ko and John J Pellegrino. Determination of osmotic pressure and fouling resistance and their effects of performance of ultrafiltration membranes. *Journal of Membrane Science*, 74(1-2):141–157, 1992.

[38] Gerardo Lafferriere, George J Pappas, and Shankar Sastry. O-minimal hybrid systems. *Mathematics of control, signals and systems*, 13:1–21, 2000.

[39] Kim G Larsen, Paul Pettersson, and Wang Yi. Uppaal in a nutshell. *International journal on software tools for technology transfer*, 1:134–152, 1997.

[40] Bareera Maryam and Hanife Büyükgüngör. Wastewater reclamation and reuse trends in turkey: Opportunities and challenges. *Journal of Water Process Engineering*, 30:100501, 2019.

[41] Carolina Quezada, Humberto Estay, Alfredo Cassano, Elizabeth Troncoso, and René Ruby-Figueroa. Prediction of permeate flux in ultrafiltration processes: A review of modeling approaches. *Membranes*, 11(5):368, 2021.

[42] Alejandro Ruiz-García, Noemi Melián-Martel, and Ignacio Nuez. Short review on predicting fouling in ro desalination. *Membranes*, 7(4):62, 2017.

[43] Alyson Sagle and Benny Freeman. Fundamentals of membranes for water treatment. *The future of desalination in Texas*, 2(363):137, 2004.

[44] Philippe Schnoebelen. The complexity of temporal logic model checking. *Advances in modal logic*, 4(393-436):35, 2002.

[45] Keith Scott and Ronald Hughes. *Industrial membrane separation technology*. Springer Science & Business Media, 1996.

[46] Aalborg University. Ai for the people. `https://www.ai.aau.dk/about/`. [Online; accessed 25-June-2024].

[47] Uppsala University and Aalborg University. Uppaal manual. `https://docs.uppaal.org/language-reference/system-description/templates/edges/`. [Online; accessed 24-June-2024].

[48] Shin-Ling Wee, Ching-Thian Tye, and Subhash Bhatia. Membrane separation process—pervaporation through zeolite membrane. *Separation and Purification Technology*, 63(3):500–516, 2008.

[49] JG Wijmans, S Nakao, JWA Van Den Berg, FR Troelstra, and CA Smolders. Hydrodynamic resistance of concentration polarization boundary layers in ultrafiltration. *Journal of Membrane Science*, 22(1):117–135, 1985.

[50] Jun Jie Wu. Improving membrane filtration performance through time series analysis. *Discover Chemical Engineering*, 1(1):7, 2021.

[51] Zi Yang, Yi Zhou, Zhiyuan Feng, Xiaobo Rui, Tong Zhang, and Zhien Zhang. A review on reverse osmosis and nanofiltration membranes for water purification. *Polymers*, 11(8):1252, 2019.

[52] Håkan LS Younes, Marta Kwiatkowska, Gethin Norman, and David Parker. Numerical vs. statistical probabilistic model checking. *International Journal on Software Tools for Technology Transfer*, 8:216–228, 2006.

[53] Håkan LS Younes and Reid G Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In *Computer Aided Verification: 14th International Conference, CAV 2002 Copenhagen, Denmark, July 27–31, 2002 Proceedings 14*, pages 223–235. Springer, 2002.