# GUILHERME ANTÓNIO DUARTE FONSECA 

Bachelor in Computer Science and Engineering

## ON FORGETTING RELATIONS IN RELATIONAL DATABASES

MASTER IN COMPUTER SCIENCE AND ENGINEERING
NOVA University Lisbon
November, 2021

# ON FORGETTING RELATIONS <br> IN RELATIONAL DATABASES 

# GUILHERME ANTÓNIO DUARTE FONSECA <br> Bachelor in Computer Science and Engineering 

Adviser: João Alexandre Carvalho Pinheiro Leite Associate Professor, NOVA University Lisbon

Co-adviser: Ricardo João Rodrigues Gonçalves
Assistant Professor, NOVA University Lisbon

Examination Committee<br>Chair: Bernardo Parente Coutinho Fernandes Toninho<br>Assistant Professor, NOVA University Lisbon<br>Rapporteur: Vitor Manuel Beires Pinto Nogueira<br>Assistant Professor, University of Évora<br>Member: Ricardo João Rodrigues Gonçalves<br>Assistant Professor, NOVA University Lisbon

## On Forgetting Relations in Relational Databases

Copyright © Guilherme António Duarte Fonseca, NOVA School of Science and Technology, NOVA University Lisbon.
The NOVA School of Science and Technology and the NOVA University Lisbon have the right, perpetual and without geographical boundaries, to file and publish this dissertation through printed copies reproduced on paper or on digital form, or by any other means known or that may be invented, and to disseminate through scientific repositories and admit its copying and distribution for non-commercial, educational or research purposes, as long as credit is given to the author and editor.

[^0]To my mother, who always encouraged me to have a curious mind and a kind heart.

## Acknowledgements

First of all, I would like to express my most sincere gratitude to my advisors Prof. João Leite, Prof. Ricardo Gonçalves and Prof. Matthias Knorr. I am very grateful to them for taking me in at the beginning of 2019 and for being an invaluable source of help and inspiration ever since. I thank them for their teachings and mentorship over these years.

I would also like to leave a word of appreciation to all my Professors at FCT NOVA and TU Delft. Thank you for everything you taught me and for your patience.

In addition, I want to thank all colleagues and friends with whom I shared this amazing five-year journey. Particularly, my acknowledgements go to João Santos, Miguel Cardoso and Paulo Mimoso for the unforgettable memories and fruitful discussions, for every "bit" of help and, above all, for the comradeship and the laughs. I am very glad that our paths have crossed each other.

I would like to thank my family for all their love, support and encouragement. In special, my deepest thanks go to my mother, for teaching me to work hard and to value life; to my sister, for being the best friend I could have asked for; para o meu avô, por ser o meu grande companheiro de estudo (e de histórias) durante estes cinco anos (bem haja!); para a minha avó, pelo carinho e pelo apoio incansável; and to my late father, Marco, who I wished was still here to read his name on this dissertation. Thank you all for bringing out the best in me.

Finally, I thank God for all the opportunities I have been granted in life.
This work was supported by FCT project FORGET (PTDC/CCI-INF/32219/2017).
"Without forgetting it is quite impossible to live at all." (Friedrich Nietzsche)


#### Abstract

Although not usually acknowledged as such, forgetting is a crucial aspect of human reasoning. It allows us to deal with large amounts of information, pushing irrelevant details out of our consciousness so that we can focus on the essential knowledge. Motivated by its beneficial effect on the human brain, this operation has been emulated in many formalisms in the field of Knowledge Representation and Reasoning, where several approaches to forgetting have been proposed. In common, these support computer systems dealing with inaccurate or excessive information without negatively affecting the remaining knowledge. More recently, the General Data Protection Regulation's 'right to be forgotten' has given additional impetus to the study of this operation.

Surprisingly, forgetting has not yet been studied in relational databases, the most widespread technology for knowledge representation. This is a serious drawback that needs to be addressed, considering the prominence of databases in our society and the relevance of the operation in numerous knowledge processing tasks.

In this dissertation, we take the first steps to tackle this need, proposing a theoretical investigation of forgetting relations in relational databases. We start by introducing an alternative formalisation of the relational model, which includes a novel notion of equivalence between databases. Afterwards, we look further into the problem of forgetting. We formally define the general concept of a relation forgetting operator and present concrete operators, each aligned with a distinct view on the operation and thus with its unique features. Moreover, we illustrate the operators with examples inspired by realistic situations. Finally, we evaluate them. For that, we formalise in the form of properties the requirements that guided the definition of the operators and prove that they satisfy desirable properties. Ultimately, with this work, we motivate the importance of forgetting in relational databases and lay the foundations for its study.


Keywords: forgetting, relational databases, relational model, General Data Protection Regulation, right to be forgotten

## Resumo

Embora nem sempre reconhecido como tal, o esquecimento é um aspeto crucial do racioć́nio humano, pois permite-nos lidar com grandes quantidades de informação, ajudandonos a concentrar no conhecimento essencial. Motivada pelo seu efeito benéfico no cérebro humano, esta operação tem sido emulada em diversos formalismos na área da Representação do Conhecimento e Raciocínio, onde várias abordagens ao esquecimento têm sido propostas. Em comum, estas apoiam sistemas informáticos a lidar com informação imprecisa ou excessiva sem afetar negativamente o restante conhecimento. Mais recentemente, o 'direito ao esquecimento' do Regulamento Geral sobre a Proteção de Dados deu um impulso extra ao estudo desta operação.

Surpreendentemente, o esquecimento ainda não foi estudado em bases de dados relacionais, a tecnologia mais utilizada para representação de conhecimento. Este é um grave inconveniente a resolver, tendo em conta a proeminência das bases de dados na nossa sociedade e a relevância da operação em inúmeras tarefas de processamento de conhecimento.

Nesta dissertação, damos os primeiros passos no sentido de fazer frente a esta necessidade, propondo uma investigação teórica do esquecimento de relações em bases de dados relacionais. Começamos por introduzir uma formalização alternativa do modelo relacional, que inclui uma nova noção de equivalência entre bases de dados. Posteriormente, analisamos mais aprofundadamente o problema do esquecimento. Definimos formalmente o conceito geral de um operador de esquecimento de relações e apresentamos operadores concretos, cada um alinhado com uma visão distinta sobre a operação e, portanto, com as suas características únicas. Ademais, ilustramos os operadores com exemplos inspirados em situações reais. Finalmente, avaliamo-los. Para isso, formalizamos sob a forma de propriedades os requisitos que orientaram a definição dos operadores e provamos que estes satisfazem propriedades desejáveis. Em última análise, com este trabalho, motivamos a importância do esquecimento em bases de dados relacionais e estabelecemos as bases para o seu estudo.

Palavras-chave: esquecimento, bases de dados relacionais, modelo relational, Regulamento Geral sobre a Proteção de Dados, direito ao esquecimento

## Contents

List of Definitions ..... xvii
List of Algorithms ..... xix
1 Introduction ..... 1
1.1 Context and Motivation ..... 1
1.2 Problem Statement and Main Contributions ..... 6
1.3 Document Structure ..... 7
2 Related Work ..... 9
2.1 Forgetting in Propositional Logic ..... 9
2.2 Forgetting in Predicate Logic ..... 11
2.2.1 Forgetting Facts ..... 11
2.2.2 Forgetting Relations/Predicates ..... 13
2.2.3 Forgetting Individuals ..... 14
2.3 Forgetting in Description Logics ..... 15
2.4 Forgetting in Answer Set Programming ..... 17
2.5 GDPR and Data Encryption ..... 20
2.6 Schema Evolution ..... 22
3 A Model for Relational Databases ..... 25
3.1 Relation ..... 25
3.2 Functional Dependency ..... 28
3.3 Database ..... 30
3.4 Database Query ..... 31
3.5 Database Equivalence ..... 31
4 Forgetting Relations in Relational Databases ..... 41
4.1 Motivating Example ..... 41
4.2 Relation Forgetting ..... 44
4.3 Non-Transitive Forgetting ..... 45
4.4 Transitive Forgetting ..... 46
4.5 Refining Transitive Forgetting ..... 53
5 Analysis and Evaluation ..... 63
5.1 Relation Forgetting Operators ..... 63
5.2 Relation Forgetting Properties ..... 66
5.3 Summary and Discussion ..... 85
6 Conclusions ..... 91
6.1 Future Work ..... 93
Bibliography ..... 97

## List of Definitions

3.1 Definition (Attribute Domain) ..... 26
3.2 Definition (Relation Schema) ..... 26
3.3 Definition (Relation Instance) ..... 26
3.4 Definition (Relation) ..... 27
3.5 Definition (Inclusion between Relations) ..... 27
3.6 Definition (Equivalence between Relations) ..... 27
3.7 Definition (Functional Dependency Satisfaction) ..... 28
3.8 Definition (Closure of a set of Functional Dependencies) ..... 28
3.9 Definition (Transitivity Rule) ..... 28
3.10 Definition (Equivalence between FD Sets) ..... 28
3.11 Definition (Projection of a Set of Functional Dependencies in a Relation Schema) ..... 29
3.12 Definition (Exclusive Projection of a Set of Functional Dependencies in a Relation Schema) ..... 29
3.13 Definition (Relational Database) ..... 30
3.14 Definition (Schema of the Relations in a Database) ..... 30
3.15 Definition (Database Schema) ..... 30
3.16 Definition (Instance of a Database Schema) ..... 30
3.17 Definition (Query) ..... 31
3.18 Definition (Conjunctive Relational Algebra Query [AIR99]) ..... 31
3.19 Definition (Derivability with respect to $\mathcal{Q}$ between Database Schemas) . ..... 32
3.20 Definition (Derivability with respect to $\mathcal{Q}$ between Databases) ..... 34
3.21 Definition (Equivalence with respect to $\mathcal{Q}$ between Databases) ..... 36
4.1 Definition (Relation Forgetting Operator) ..... 45
4.2 Definition (Non-comparability between Relations) ..... 47
4.3 Definition (Non-total Intersecting Superset) ..... 48
4.4 Definition ( $\delta$-Exclusion of a Set of Functional Dependencies) ..... 51
5.1 Definition (Persistence (P)) ..... 67
5.2 Definition (Weakening (W)) ..... 68
5.3 Definition (Irrelevance (I)) ..... 70
5.4 Definition (Equivalence up to $\delta$ between Databases) ..... 70
5.5 Definition (No Transitivity (NT)) ..... 71
5.6 Definition (Strong Preservation of Functional Dependencies (SPFD)) ..... 72
5.7 Definition (weak Preservation of Functional Dependencies (wPFD)) ..... 73
5.8 Definition $(\psi$-Irrelevance $(\psi-\mathbf{I}))$ ..... 74
5.9 Definition (Transitivity with Deletion ( $\mathbf{T}^{-}$) ..... 75
5.10 Definition (Transitivity with Addition ( $\mathrm{T}^{+}$)) ..... 76
5.11 Definition (No Recovery of Forgotten Tuples (NRT)) ..... 78

## List of Algorithms

1 Computation of $F_{r}^{R}$ ..... 29
2 Non-transitive relation forgetting operator $\mathrm{f}_{n t}$ ..... 46
3 Computation of the $\delta$-exclusion of a set of FDs ..... 51
4 Transitive relation forgetting operator $\mathrm{f}_{t}$ ..... 52
5 Transitive relation forgetting operator $\mathrm{f}_{t^{-}}$ ..... 56
6 Transitive relation forgetting operator $\mathrm{f}_{t^{+}}$ ..... 61

## Introduction

In this first chapter, we motivate the process of forgetting both in the broader context of human reasoning and in the narrower scope of computer science. Subsequently, we define the problem we propose to address in this dissertation and highlight its main contributions. We conclude with an outline of the document structure.

### 1.1 Context and Motivation

In the human brain, forgetting is the spontaneous process in which less accessed information gradually fades into the background. This operation can be seen as a function over time since it is directly impacted by the frequency with which we actively recall a specific event or information we ought to remember. In fact, whenever we cease to engage with some particular knowledge for long periods, or even when similar, more recent learnt information interferes with our ability to recollect the original, it is only natural that those memories start to vanish from our consciousness. However, forgetting what once was known does not directly imply that such knowledge ceases to completely exist (i.e., it is forgotten forever). In the same way that, once in the background, information can be permanently removed (even if at different rates), it is also evident that, through the opposite process, it can be eventually remembered, reemerging vividly into the first plan. This proceeding may happen, for instance, through recall by association with other knowledge [Ebb85].

It happens that, to this notion of forgetting, it is generally attributed a negative connotation, strictly related to the involuntary loss of important knowledge over time. Nonetheless, from a cognitive perspective, the process of forgetting is a crucial aspect of human reasoning. It is the constant juggling of memories, from the background to the first plan and vice versa, allied to the ability to permanently forget, that helps humans deal with an excess of information, allowing us to abstract away from the irrelevant and focus on the knowledge that really matters when performing a certain task. Moreover, forgetting enables us to accommodate new information without the constant interference of older (perhaps contradictory or worse) knowledge.

Because of its negative implications, even though the concept of forgetting has been thoroughly studied in the cognitive and social sciences for long (with works such as [Ebb85] from the field of Psychology), it is undeniable that, until more recently, this operation did not receive due attention within the community of Computer Science [EK19]. This is especially striking in the discipline of Knowledge Representation and Reasoning (KRR), an important area of Artificial Intelligence (AI) which not only studies how information can be symbolically represented by computer systems but also how it can be manipulated in an automated way by reasoning programs so that it is possible to derive implicit knowledge [BL04].

More recently, with the increasing acknowledgement of the relevance of this operation in the field of KRR, consequence, to a great extent, of the information revolution, the last decades have seen an uprising in the investigation of forgetting in many formalisms of knowledge representation. Among them, forgetting has been studied in propositional logic, predicate logic (i.e., first order logic), description logics and non-monotonic logic (for instance, in Answer Set Programming (ASP)).

Naturally, this spike of interest stems from the capacity of this procedure to handle irrelevant or inaccurate information, proven by the beneficial effect that forgetting has on the human mind. Nonetheless, even if the purpose of the operation is essentially the same in both domains, contrary to what happens in the human brain, where this process is mainly unintentional, forgetting in KRR is always intentional. In practice, this means that the act of forgetting is deliberately triggered in order to solve a particular problem or answer a specific need. In that respect, forgetting can be extremely helpful in many situations. For example, with the ever-growing necessity of dealing with large amounts of information, this procedure can be intentionally used to simplify the knowledge acquired from a certain world by excluding irrelevant details. Similarly, forgetting can be applied to improve the declarativity of a knowledge base $(\mathrm{KB})^{1}$, eliminating auxiliary concepts that have no particular meaning in the world under consideration and temporary elements that are no longer necessary. In addition, with the increasing concern about data protection and privacy, recognised in the recent General Data Protection Regulation $(G D P R)^{2}$, this operation can also be employed to deal with data collected unlawfully or to implement the GDPR's 'right to be forgotten'. In that regard, the European Parliament and the Council of the European Union (EU) emitted on the 26th of April 2016 the Regulation (EU) 679/2016, known as the GDPR, which recognises the right to erasure (or 'the right to be forgotten') ${ }^{3}$. Under this right, European citizens can request organisations for their personal information to be forgotten (i.e., removed from the internet or any business processes) under certain conditions.

[^1]With that said, even though a goal of forgetting from a KRR perspective is to exclude information from a knowledge base, this procedure is not limited to knowledge removal, as there is more to it than the simple omission of information. On the other hand, forgetting is strongly related to other knowledge representation concepts like irrelevance and equivalence, which further justify its importance in a broad range of reasoning tasks [LR94]. Therefore, before we can continue discussing this operation in KRR, it is essential to establish as of now, even if informally, a definition that captures its main ideas in order to better distinguish it from other reasoning techniques.

To that end, it is important to consider that there are, in fact, two major interpretations regarding the operation of forgetting in the literature. The first, distinguished as type 1 , focuses on eliminating certain parts of the vocabulary used to represent the knowledge base, while preserving as much information as possible. The second, known as type 2, looks at forgetting through a different standpoint. It attempts at directly changing the KB so that the knowledge to be forgotten can no longer be inferred. In the context of KRR, the first view on forgetting is generally more predominant, whereas the second has received less attention ${ }^{4}$.

For this reason, in this dissertation, we focus on forgetting operations of type 1. In that case, perhaps a more intuitive definition of this type of forgetting can be found in [Lei17] for Answer Set Programming, and can be generalised to other knowledge representation formalisms. Intuitively, forgetting is an operation that allows the removal, from a knowledge base, of a piece of information no longer deemed relevant in such a way that both direct and indirect relationships between the remaining knowledge are preserved.

To illustrate this definition, let us consider a simple example, still in ASP ${ }^{5}$.
Example 1.1. Consider the following ASP knowledge base (or program)

$$
K B=\{a \leftarrow c . \quad c \leftarrow b . \quad d \leftarrow \text { not } e . \quad b \leftarrow .\} .
$$

The program $K B$ has three rules, which can be read as "if $c$ is true then $a$ must be true", "if $b$ is true then $c$ must be true", " $d$ is true unless $e$ is true", and the fact $b$. If we wish to forget variable $c$ from the program $K B$, denoting the result as $\operatorname{forget}(K B, c)$, then we have to keep the rule $d \leftarrow n o t e$ and the fact $b$, as hinted by the definition above since they are independent of variable $c$. What is more, the logical consequence $a \leftarrow b$, which is indirectly present in the program $K B$ through $c$, and can be read as "if $b$ is true then $a$ must be true", should be preserved. With that in mind, the result of the operation forget ( $K B, c$ ) can be the program

$$
K B^{\prime}=\{a \leftarrow b . \quad d \leftarrow \text { not } e . \quad b \leftarrow .\},
$$

which ensures the removal of variable $c$ and leaves the logical consequences (both direct and indirect) between the remaining variables intact. In that respect, note that $K B^{\prime}$ was

[^2]not obtained by merely removing $c$, nor removing the rules with $c$. In order to preserve indirect relationships, we had to construct a new rule $a \leftarrow b$.

As can be seen in the example, just like in the cognitive view, forgetting in KRR is generally done in such a way that its consequences on the knowledge that remains are minimal. That is, the operation usually seeks to preserve all the relationships among the information that persists, even those that were dependent on what was forgotten. Clearly, this explicit intention in preserving the overall meaning of the KB distinguishes forgetting from a mere remove operation, where knowledge is simply discarded without any consideration for the repercussions that its deletion might have on the remainder.

However, despite the simplicity of the example above, forgetting is usually a difficult task. As a matter of fact, although there exist definitive solutions to this problem in some formalisms (as is the case, e.g., of propositional logic and predicate logic), it still remains largely an open problem in many others (such as, e.g., in some description logics and ASP). There are several reasons we can pinpoint to justify the complexity of this operation. For instance, sometimes it happens that the elements in a KB are so tightly coupled that it is impossible to remove some of them without severely affecting the connections therein. Furthermore, one other reason that can be attributed to this difficulty is related to the fact that, while the result of forgetting can be expressed at a syntactic level, as depicted in Example 1.1, it can also be characterised at the semantic level. In particular, the latter focuses on the models (i.e., the interpretations ${ }^{6}$ ) that the KB should have after the operation. This brings additional challenges in the sense that sometimes it is impossible to express syntactically in the language of the original KB the result of semantic forgetting (in fact, this problem may even be undecidable) [EK19]. A third reason, which in reality is more aligned with the difficulties in ASP, is due to the fact that, in specific situations, it is not possible to maintain certain desirable properties, generally related to some notion of equivalence between the original and resulting KBs, upon the result of forgetting [Gon+20]. In those circumstances, it is necessary to settle for other, probably less desired results.

As such, in spite of the informal definition presented, for the formalisms where the problem of forgetting has not yet been completely solved, there is no single formal definition for this concept, much less a unique operator of forgetting. This happens because, in these cases, the operation is generally contemplated in light of distinct perspectives, each prompted by a specific set of applications and their requirements. Therefore, although they are based on a common intuition, the different approaches end up specifying disparate characteristics, a direct result of the particular tasks they were designed to accomplish.

In that respect, forgetting has proved to be a pivotal operation in numerous fields of

[^3]AI, where its applications are manifold. For instance, this operation can be used to optimise the efficiency of reasoning systems, made possible by the suppression of irrelevant, redundant or even inconsistent knowledge [EK19; LLM03; Wan+08]; restructure and decompose a knowledge base into smaller parts (e.g., Wang et al. [Wan+08] picture a scenario where the contents of a large knowledge base about Medicine, describing a myriad of diseases and treatments, have to be split into smaller parts with respect to specialties such as Oncology and Odontology so that, by discarding the unrelated concepts, it is easier to use specific portions of the KB); optimisation of query answering by distilling from a KB only what is relevant to the query [Del14; LLM03]; knowledge update and belief revision [BZ05; LLM03]; planning, reasoning about action and cognitive robotics (e.g., the work by Lin and Reiter in [LR94] was motivated by the need for an autonomous robot to forget about her past actions without negatively impacting the future ones); conflict resolution [ZF06], among many others.

Forgetting has thus been extensively studied in numerous knowledge representation formalisms, with important use cases and applications, as witnessed by the examples above. Nevertheless, after all these investigations, it is surprising to see that the most widespread technology for knowledge representation - relational databases - has not yet been the subject of research on this topic. In effect, in the context of relational databases, forgetting could as well be a valuable tool, serving similar purposes as those discussed for the other formalisms. For example, it could be useful to simplify the state of a database by removing auxiliary or temporary tables that are no longer necessary or were incorrectly defined. At the same time, it would preserve the relationships among the remaining data that were established by the forgotten tables. Ultimately, this could lead to improvements in the database performance, which would translate into the optimisation of query answering and speed-up data processing, without compromising the overall meaning of the data. In the same way, forgetting could be used to handle physical storage limitations and to restructure or summarise parts of the database in order to merge or reuse them in other systems. In addition, it could also be employed to deal with the GDPR and to implement its 'right to be forgotten'.

The latter, in fact, is a particularly relevant application for forgetting in relational databases, considering that organisations mostly rely on them as their de facto data storage systems and that there is a growing interest in properly addressing data protection and privacy issues. Certainly, even though organisations currently resort to techniques such as pseudonymisation or anonymisation ${ }^{7}$, a forgetting operation in relational databases

[^4]in line with forgetting in KRR could be an additional solution to meet the GDPR's demands at a practical level.

### 1.2 Problem Statement and Main Contributions

In view of the points just raised, in this dissertation we propose an investigation on forgetting in the context of relational databases where, unlike other knowledge representation formalisms, no attention has been devoted to this problem.

More specifically, the primary goals of this dissertation are to develop a theory of forgetting relations (i.e., tables) in relational databases and to define concrete operators for forgetting relations that address requirements of the 'right to be forgotten' and thus facilitate its implementation in practical systems. To that end, in addition to the implications of forgetting as a procedure to strictly deal with excessive, imprecise or no longer useful information, we will have to take into account the ramifications of dealing with data no longer allowed to be held due to legal and privacy concerns.

In order to achieve these goals, we start by identifying requirements for forgetting both in the GDPR and the existing academic literature on forgetting. Only then can we formalise operators that implement forgetting both at the level of the schema and the data of the database, while satisfying relevant subsets of the requirements compiled.

With that said, the main contributions of this dissertation are the following:

1. An alternative formalisation of the relational model specifically tailored to address the problem of forgetting in relational databases;
2. A novel notion of equivalence between databases based on the comparison of the databases' ability to represent information;
3. The concept of forgetting in relational databases, including the formalisation of the general notion of a relation forgetting operator;
4. The definition of four concrete operators of relation forgetting, each aligned with a different view on the operation, which in turn was motivated by a specific set of requirements found in pragmatic use cases;
5. The formalisation in the form of properties interesting requirements for forgetting found in real-world scenarios, the GDPR and the related literature, including the ones that motivated our operators;
6. An analysis and analytical evaluation of the operators with respect to the properties defined, proving that, besides being correctly defined, the operators obey desirable sets of such properties.
[^5]
### 1.3 Document Structure

The remainder of this dissertation is structured as follows:

Chapter 2 - Related Work: In this chapter, we start by overviewing the literature on the operation of forgetting for different knowledge representation formalisms. Namely, we focus on propositional logic, predicate logic, description logics and Answer Set Programming. Afterwards, we discuss existing methods to deal with the GDPR's 'right to be forgotten' in relational databases and comment on the relationship between the field of schema evolution and the operation of forgetting.

Chapter 3 - A Model for Relational Databases: In this chapter, we define the main concepts that will be used to formalise our theory of forgetting relations in relational databases.

Chapter 4 - Forgetting Relations in Relational Databases: In this chapter, we investigate the operation of forgetting relations in relational databases. We start by presenting a practical example that justifies the need for forgetting relations and motivates different approaches to do so. Subsequently, we formally present the definition of a relation forgetting operator. Then, we introduce several operators of forgetting and illustrate them with a few examples. To do so, we divide the study of the operators in light of two primary interpretations of the operation.

Chapter 5 - Analysis and Evaluation: In this chapter, we analyse our operators of forgetting. In the first part, we prove that the operators are well defined, i.e., they are in accordance with the notion of a relation forgetting operator introduced in the previous chapter. In the second part, we evaluate analytically the operators. To that end, we introduce several formal properties, each corresponding to a self-contained requirement for forgetting that guided the definition of the operators, and show which properties are satisfied by which operators. Finally, in the last part, we reflect on the work covered in this dissertation, summarise the main results and propose enhancements to our operators.

Chapter 6 - Conclusions: In this chapter, we conclude the dissertation by presenting a general overview of the main contributions and suggesting directions for future developments of our theory.

## Related Work

The operation of forgetting has been studied in different logic-based formalisms within the Knowledge Representation and Reasoning community. Notably, one characteristic is common to the motivations of such work: reasoning should focus on the important knowledge. With that in mind, the first part of this chapter has the purpose of outlining the different approaches to forgetting discussed in the literature for various knowledge representation formalisms. First, we will focus on propositional logic and predicate logic, where we assume the reader has familiarity with these formalisms. Next, we discuss forgetting in description logics and non-monotonic logic (particularly Answer Set Programming). Finally, we step away from forgetting in KRR and move to the domain of relational databases, where we start by focusing on data encryption as a technique to deal with the GDPR's 'right to be forgotten', comparing it with a desirable operator of forgetting; and conclude with a brief overview on schema evolution and how it relates to forgetting.

### 2.1 Forgetting in Propositional Logic

The operation of forgetting in propositional logic goes back to the seminal work of George Boole in [Boo54]. In this work, Boole addressed the exclusion of middle terms (or variable elimination) as a way of omitting undefined or irrelevant variables from propositional formulas. Specifically, the result of forgetting a variable from any given formula $\phi$ is obtained by replacing in $\phi$ the variable to be forgotten with $\top=$ true, and with $\perp=$ false (i.e., its possible valuations), and joining the resulting formulas with the logical disjunction operator $(\mathrm{V})$. To show this procedure, in the example below, we assume a knowledge base $K B$ where set of formulas were combined into a single formula $\phi$ with the operator of logical conjunction $(\wedge)$.

Example 2.1 (cf. [EK19]). Consider the knowledge base $K B=\{\phi\}$, where

$$
\phi=(\text { red_wine } \vee \text { white_wine }) \wedge(\text { fish } \rightarrow \text { white_wine }) \wedge(\text { beef } \rightarrow \text { red_wine }) .
$$

This knowledge base describes knowledge about wine and foods. More specifically, the first formula in $\phi$ means "either white wine or red wine should be drunk", the second formula means "fish must be accompanied by white wine", and the last "beef must be accompanied by red wine". Imagine we want to forget about $f$ ish in $K B$. Then, by the definition above, the resulting knowledge base would be (with red_wine, white_wine, $f$ ish and beef abbreviated to $r, w, f$ and $b$, respectively)

$$
\begin{aligned}
K B^{\prime} & =\{[(r \vee w) \wedge(T \rightarrow w) \wedge(b \rightarrow r)] \vee[(r \vee w) \wedge(\perp \rightarrow w) \wedge(b \rightarrow r)]\} \\
& =\{[(r \vee w) \wedge w \wedge(b \rightarrow r)] \vee[(r \vee w) \wedge(b \rightarrow r)]\} \\
& =\{(r \vee w) \wedge(b \rightarrow r)\},
\end{aligned}
$$

where " $=$ " means logical equivalence.
This notion can be extended to knowledge bases consisting of finite (or even infinite) sets of formulas [EK19]. In that case, a new knowledge base $K B^{\prime}$ results from forgetting a variable in the original knowledge base $K B$. This is done by replacing, for all possible pairs of formulas, the variable to be forgotten with true in the first formula, and false in the second, and joining both results. More formally, and assuming that we want to forget variable $p$ from $K B$, then $K B^{\prime}=\operatorname{forget}(K B, p)=\left\{\phi_{p}^{+} \vee \psi_{p}^{-} \mid \phi, \psi \in K B\right\}$, where $\phi_{p}^{+}$is $\phi$ with $p$ replaced by T , and $\psi_{p}^{-}$corresponds to $\psi$ with $p$ replaced by $\perp$.

Even though the aforementioned definition happens at the syntactic level, forgetting in propositional logic can also be characterised semantically. For such characterisation, the key idea is that the models of $K B$ and $K B^{\prime}$ agree on every variable except possibly on the variable to be forgotten. In particular, this can be done by choosing whether or not to keep the original signature (a.k.a. vocabulary), which brings us to two distinct definitions regarding the models of $K B^{\prime}$. The first (1) corresponds to keeping the original signature and adding to the set of original models all the models that can be obtained by considering the variable to be forgotten as true and as false (i.e., by removing and adding the variable to be forgotten from each original model). The second approach (2) considers as the new signature of $K B^{\prime}$ the signature of $K B$ except the variable to be forgotten, eliminating its occurrence from all the original models [EK19].
Example 2.2 (cf. [EK19]). Consider the knowledge base $K B$ from Example 2.1, constructed on signature $\Sigma$. The set of all models of $K B$ over $\Sigma$, denoted by $\operatorname{Mod}_{\Sigma}(K B)$ corresponds to the set $\{\{r\},\{w\},\{r, w\},\{b, r\},\{b, r, w\},\{f, w\},\{f, w, r\},\{f, b, w, r\}\}$. The semantic result of forgetting about $f$ in $K B$ is given by
(1) $\operatorname{Mod}_{\Sigma}(f \operatorname{orget}(K B, f))=\operatorname{Mod}(K B) \cup\{\{f, r\},\{f, b, r\}\}$,
(2) $\operatorname{Mod}_{\Sigma \backslash\{f\}}(\operatorname{forget}(K B, f))=\{\{r\},\{w\},\{r, w\},\{b, r\},\{b, r, w\}\}$,
where (1) and (2) correspond to the respective alternatives.
The definition of forgetting can be further extended to a set of variables by iterative application of the operator $\operatorname{forget}(K B, p)$. In that case, it holds that the result
obtained is independent of the order in which the iteration is processed (a property known as order independence) [LR94]. Moreover, Lin and Reiter show that this definition of forgetting obeys a set of other interesting properties. For instance, at the semantic level, the operator forget preserves all models of $K B$ (over-approximation), and all consequences not mentioning any forgotten variable are kept (consequence invariance). It follows that, if a variable to be forgotten $p$ does not occur in a formula $\phi \in K B$, then we can leave the formula untouched. More formally, for such cases we have that $f$ orget $(K B, p)=\operatorname{forget}(K B \backslash\{\phi\}, p) \cup\{\phi\})[$ EK19].

Furthermore, there are alternative definitions to forgetting that explore specific formula arrangements in knowledge bases. For example, the work in [Del14] deals with KBs in clausal form (or conjunctive normal form), meaning that all formulas $\phi$ are disjunctions of literals which, in their turn, can be variables $p$ or their negations $\neg p$. The result of forgetting according to this definition is logically equivalent to that of the Boole definition.

Whereas the operations above aim at forgetting a variable (or set of variables) from a knowledge base, the work by Lang et al. [LLM03] proposes an operation to forget literals, distinguishing any atom $p$ from its negation $\neg p$. This means that, unlike those definitions, when forgetting about a literal, it may happen that the resulting knowledge base still mentions the variable, although only in the form of the literal contrary to the one forgotten. Literal forgetting can, as well, be extended to multiple literals by simple iteration, where the order in which they are forgotten does not matter. Moreover, forgetting a variable with the operator forget is equivalent to forgetting both its positive and negative literal forms through literal forgetting.

Finally, note that all the forgetting operations mentioned previously can be, indeed, categorised as type 1 with respect to the classification of forgetting presented in the Introduction. This is so because there is an explicit intention in reducing the signature of the KB, rather than changing the KB per se by removing specific formulas so that they are no longer inferable.

### 2.2 Forgetting in Predicate Logic

Similarly to propositional logic, forgetting was thoroughly contemplated in predicate logic (first-order logic (FOL)), both at the level of facts (e.g., forget that John is a student), as well as relations/predicates (forget the relation student) and individuals (forget John). In this section, we briefly comment on forgetting these concepts.

### 2.2.1 Forgetting Facts

When it comes to the operation of forgetting facts $p=P(\mathbf{t})$, where $\mathbf{t}=t_{1}, \ldots, t_{k}$ is a tuple of terms, a first approach could be to consider the definition of forgetting in propositional logic as a special case, such that each variable $p$ corresponds to a 0 -ary predicate $P$.

However, this approach will only work for very specific cases. In the presence of existential $(\exists)$ and/or universal $(\forall)$ quantification, $\mathbf{t}$ may be accessed indirectly in the valuation of atoms over $P$, and thus syntactically substituting $P(\mathbf{t})$ by true resp. false will not guarantee the expected semantic result. To exemplify this situation consider the formula $\phi=\forall x . P(x)$. If we wish to forget $P(a)$ from $\phi$ by replacing it with true resp. false, it happens that the resulting formula will yield no semantic differences with respect to the original since $P(a)$ does not occur in $\phi$. In this case, the expected result of forgetting $P(a)$ in $\phi$ would be having models in which $P(x)$ is true for all elements $x$ different from $a$, while the value of $P$ should remain open for variable $a$ [EK19].

Therefore, in order to deal with quantifiers, it is necessary to introduce a syntactic transformation to $\phi$, changing it into a logically equivalent formula, as described in [LR94]. This can be straightforwardly done by replacing every occurrence of the form $P\left(\mathbf{t}^{\prime}\right)$ in $\phi$ by $\left[\mathbf{t}^{\prime}=\mathbf{t} \wedge P(\mathbf{t})\right] \vee\left[\mathbf{t}^{\prime} \neq \mathbf{t} \wedge P\left(\mathbf{t}^{\prime}\right)\right]$, where $\mathbf{t}^{\prime}=\mathbf{t}$ is a shorthand for $t_{1}^{\prime}=t_{1} \wedge \ldots \wedge t_{k}^{\prime}=t_{k}$. We denote the result of this transformation by $\phi[p]$.

Now, assuming a knowledge base $K B=\{\phi\}$ and a variable-free fact $p=P(\mathbf{t})$, the definition of forgetting $p$ in $K B$ is given by $\operatorname{forget}(K B, p)=\left\{\phi_{p}^{+} \vee \phi_{p}^{-}\right\}$, where $\phi_{p}^{+}$(resp. $\phi_{p}^{-}$) corresponds to the result of replacing $p$ by $T=$ true (resp. $\perp=$ false) in $\phi[p]$ [LR94].
Example 2.3 ([LR94]). Consider the knowledge base $K B=\{\phi\}$, where

$$
\phi=\operatorname{student}(J o h n) \vee \operatorname{student}(J o e) \vee \text { teacher }(\text { John }) .
$$

Suppose we want to forget about fact $p=\operatorname{student}(J o h n)$. First, we transform $\phi$ into $\phi[$ student (John)], obtaining

$$
\begin{aligned}
\phi[\operatorname{student}(J o h n)] & =[(J o h n=J o h n \wedge \operatorname{student}(J o h n)) \vee(J o h n \neq J o h n \wedge \operatorname{student}(J o h n))] \\
& \vee[(J o e=J o h n \wedge \operatorname{student}(J o h n)) \vee(J o e \neq J o h n \wedge \operatorname{student}(J o e))] \\
& \vee \text { teacher }(J o h n) .
\end{aligned}
$$

Then,

$$
\begin{aligned}
\operatorname{forget}(K B, \text { student }(\text { John })) & =\left\{\phi_{\text {student }(\text { ohn })}^{+} \vee \phi_{\text {student }(\text { John })}^{-}\right\} \\
& =\{([(T \wedge T) \vee(\perp \wedge T)] \vee[(\perp \wedge T) \vee(T \wedge \text { student }(\text { Joe }))] \vee \text { teacher }(\text { John })) \\
& \vee([(T \wedge \perp) \vee(\perp \wedge \perp)] \vee[(\perp \wedge \perp) \vee(T \wedge \text { student }(\text { Joe }))] \vee \text { teacher }(\text { John }))\} \\
& =\{\top \vee(\text { student }(\text { Joe }) \vee \text { teacher }(\text { (ohn }))\} \\
& =\{\top\} .
\end{aligned}
$$

Instead, if we consider $\phi=\exists x$.student $(x)$, then forgetting student (John) from $K B$ results in

$$
\begin{aligned}
\phi[\operatorname{student}(J o h n)] & =\exists x \cdot[x=\operatorname{John} \wedge \operatorname{student}(\operatorname{Joh} n)] \vee[x \neq \operatorname{John} \wedge \operatorname{student}(x)] \\
\operatorname{forget}(\text { KB,student }(\text { John })) & =\{\exists x \cdot[x=\operatorname{John} \wedge T] \vee[x \neq \operatorname{John} \wedge \operatorname{student}(x)] \\
& \vee \exists x \cdot[x=\operatorname{John} \wedge \perp] \vee[x \neq \operatorname{John} \wedge \operatorname{student}(x)]\} \\
& =\{\mathrm{T}\} .
\end{aligned}
$$

Finally, if $\phi=\forall x \cdot \operatorname{student}(x)$ and we still want to forget about student(John) in $K B$, then the result is

$$
\begin{aligned}
\phi[\operatorname{student}(J o h n)] & =\forall x \cdot[x=\operatorname{John} \wedge \operatorname{student}(J o h n)] \vee[x \neq \operatorname{John} \wedge \operatorname{student}(x)] \\
\operatorname{forget}(K B, \operatorname{student}(J o h n)) & =\{\forall x \cdot[x=\operatorname{John} \wedge T] \vee[x \neq \operatorname{John} \wedge \operatorname{student}(x)] \\
& \vee \forall x \cdot[x=\operatorname{John} \wedge \perp] \vee[x \neq \operatorname{John} \wedge \operatorname{student}(x)]\} \\
& =\{\forall x \cdot[x \neq \operatorname{John} \rightarrow \operatorname{student}(x)] \vee \forall x \cdot[x \neq \operatorname{John} \wedge \operatorname{student}(x)]\} \\
& =\{\forall x \cdot[x \neq \operatorname{John} \rightarrow \operatorname{student}(x)]\} .
\end{aligned}
$$

Note that this is precisely the scenario mentioned at the beginning of this section: without the transformation of $\phi$ into $\phi[p]$ before substituting $p$ by true resp. false, it would not be possible to obtain the discussed result.

Moreover, analogously to the propositional case, Lin and Reiter show that this definition of forgetting can be extended to multiple atoms by applying an iterative process. Again, the order of the atoms is irrelevant to the final result (i.e., the forget operator commutes).

Regarding the semantic characterisation of this operator, the principle is that the models of the result of forgetting fully agree with the original models except possibly on the truth assignment of the atoms to be forgotten.

In addition to the property order independence, the operator holds, for instance, the property over-approximation (all original models are kept although new ones can be added), which implies that the result of forgetting is weaker than the original knowledge base. Furthermore, the result of forgetting about a sequence of atoms always exists and does not affect other sentences "irrelevant" to those atoms [LR94].

### 2.2.2 Forgetting Relations/Predicates

In what concerns the operation of forgetting relations, it can be semantically characterised in a similar way to the previous case: the models of the knowledge base after forgetting a relation $P$ agree on everything with the original models except possibly for the valuation of $P$ [LR94].

Furthermore, Lin and Reiter introduce a syntactic definition of forgetting capturing the characterisation above. However, this definition is presented in second-order logic (SOL) in order to use quantification over relation symbols: given a knowledge base $K B=\{\phi\}$ and a relation variable $R$ with the same arity as $P$, we have $\operatorname{forget}(K B, P)=$ $\{\exists R . \phi(P / R)\}$, where $\phi(P / R)$ is the result of replacing every occurrence of $P$ in $\phi$ by $R$.

Example 2.4 ([LR94]). Consider the knowledge base

$$
K B=\operatorname{student}(J o h n) \vee \operatorname{student}(J o e) \vee \text { teacher }(J o h n) .
$$

The result of forgetting about $P=$ student in $K B$ according to the syntactic definition above is

$$
\text { forget }(K B, \text { student })=\{\exists R \cdot R(\text { John }) \vee R(\text { Joe }) \vee \text { teacher }(\text { John })\},
$$

which is equivalent to T since there exists a model in which $R($ John $)$ is true.
Alternatively, if $K B=\{(\operatorname{stu} \operatorname{dent}(\operatorname{John}) \vee \operatorname{stu} \operatorname{dent}(J o e)) \wedge$ teacher $(\operatorname{John})\}$, then by forgetting student we obtain

$$
\begin{aligned}
\text { forget }(\text { K } B, \text { student }) & =\{\exists R .(R(\text { John }) \vee R(\text { Joe })) \vee \text { teacher }(\text { John })\} \\
& =\{\text { teacher }(\text { John })\} .
\end{aligned}
$$

In the example above, the result of forgetting relation student in both knowledge bases is first-order definable. However, as noted by Lin and Reiter, this is not always the case. Thus, in those situations, the question becomes whether it is possible to transform the resulting SOL formula into an equivalent FOL formula. Notably, Doherty et al. [DLS01] show in which conditions second-order quantifiers can be eliminated, making it possible to express forgetting in FOL.

Lastly, the operation forget can also be extended to a sequence of predicates, being that the resulting knowledge base is independent of the order in which the predicates are forgotten. As a matter of fact, similar properties to the ones of forgetting about a fact hold as well when it comes to forgetting relations (e.g., order independence, over-approximation and a form of irrelevance) [LR94].

### 2.2.3 Forgetting Individuals

As for the operation of forgetting individuals (which are represented by constant symbols in first-order formulas), a definition is proposed in [Del14] as follows. For a knowledge base $K B=\{\phi\}$ and a constant symbol $c, f \operatorname{orget}(K B, c)=\left(K B \backslash K B_{c}\right) \cup\left\{\exists x . \bigwedge_{\phi \in K B_{c}} \phi[c / x]\right\}$, where $K B_{c}$ denotes the set of all sentences in $K B$ where $c$ occurs, $x$ is a variable not appearing in $K B$ and $\phi[c / x]$ is the result of replacing $c$ by $x$ in $\phi$. Regarding this definition, all consequences of $K B$ not mentioning $c$ are kept, while no new consequences are introduced [Del14].

An illustration of this notion is given by the following example.
Example 2.5 ([EK19]). Consider the knowledge base

$$
K B=\{\operatorname{student}(J o h n) \vee \operatorname{student}(J o e) \vee \text { teacher }(\text { John })\} .
$$

The result of forgetting about John in $K B$ is

$$
\text { forget }(K B, J o h n)=\{\exists x . \operatorname{student}(x) \vee \text { student }(J o e) \vee \text { teacher }(x)\} .
$$

As observed in [EK19], this operation of forgetting can be seen as a process of anonymisation in the sense that it is no longer possible to discern which individual has certain properties. In fact, at a first glance, one could try to replace $c$ with a fresh constant symbol, instead of the construct above. However, by following this approach, it will become easier to identify which forgetting operations about individuals were previously done in the knowledge base, which is not always desirable.

### 2.3 Forgetting in Description Logics

Description logics (DLs) are a family of knowledge representation formalisms that amount to FOL fragments capable of effective reasoning. These languages are expressed in a restricted yet convenient syntax that differs from the FOL syntax and are equipped with a precise formal semantics. Moreover, contrary to FOL, DLs are usually decidable (i.e., guarantee termination in finite time both for positive and negative answers) [Baa+03].

A DL signature consists of individuals (constants), concepts (classes that denote sets of individuals) and roles (binary relations on individuals). Furthermore, a DL knowledge base $K B$ comprises a TBox $\mathcal{T}$ (terminological box) and an ABox $\mathcal{A}$ (assertional box) such that $K B=\langle\mathcal{T}, \mathcal{A}\rangle$. The TBox describes knowledge about concepts and roles. Concretely, it is composed of a set of inclusion axioms of the forms $C \sqsubseteq D$ (concept inclusion) and $R \sqsubseteq S$ (role inclusion) ${ }^{1}$, where $C$ and $D$ are concept expressions, and $R$ and $S$ are role expressions. On the other hand, the ABox is a set of assertions of the forms $A(a)$ (concept membership) and $P(a, b)$ (role membership), where $a$ and $b$ are individuals, $A$ is a concept and $P$ is a role. Intuitively, an ABox specifies properties of individuals occurring in the application domain.

Description languages are inherently characterised by the constructors they provide. These constructors define in a precise way the allowed concept (and sometimes role) expressions within the language. Accordingly, different languages have different expressive powers and reasoning complexities such that, the more expressive the language is, the harder the reasoning. Overall, this tradeoff makes DLs suitable for a wide range of applications and reasoning tasks [Baa+03]. For instance, description logics can be used to express conceptual models of a database, integrate several data sources, or even formulate queries.

Additionally, DLs are particularly useful for writing ontologies. These, in turn, can be used to facilitate access to data repositories [Xia+18]. Not surprisingly, the World Wide Web Consortium ${ }^{2}$ recommends that languages for describing ontologies in the Web should be based on DLs. One such language is OWL2 ${ }^{3}$, a variant of the description logic $\mathcal{S R O} \mathcal{I} \mathcal{Q}^{(\mathcal{D})}$. However, perhaps the most known DL, which also happens to be the smallest propositionally closed description logic is $\mathcal{A L C}$ (attributive language with complement). Even so, in this section, we zoom in on the literature of forgetting for the DL-Lite family of description logics, whose languages are particularly attractive due to their expressive power while still having polynomial time reasoning algorithms in the worst case [Cal+07]. In specific, we turn our attention to the simplest language of this family, DL-Lite ${ }_{\text {core }}$.

Example 2.6 (cf. [Wan+08]). Consider the following DL-Lite ${ }_{\text {core }}$ knowledge base $K B=$ $\langle\mathcal{T}, \mathcal{A}\rangle$, which describes a simple library domain, followed by the FOL translation for each

[^6]axiom respectively assertion.

|  | DL-Lite ${ }_{\text {core }} K B$ | FOL translation |
| :---: | :---: | :---: |
| $\mathcal{T}$ | $\exists$ OnLoanTo LibItem | $\forall x(\exists y$.OnLoanTo $(x, y) \rightarrow \operatorname{LibItem}(x))$ |
|  | $\exists$ OnLoanTo- $\preceq$ Member | $\forall x(\exists y$.OnLoanTo $(y, x) \rightarrow$ Member $(x))$ |
|  | Member $\sqsubseteq$ Person | $\forall x(\operatorname{Member}(x) \rightarrow \operatorname{Person}(x))$ |
|  | Visitor $\sqsubseteq P e r s o n$ | $\forall x(\operatorname{Visitor}(x) \rightarrow \operatorname{Person}(x))$ |
|  | Visitor $\sqsubseteq \neg$ Member | $\forall x(\operatorname{Visitor}(x) \rightarrow \neg \operatorname{Member}(x))$ |
| $\mathcal{A}$ | LibItem(SWPrinter) | LibItem(S WPrinter) |
|  | onLoanTo(DLHandBook,Jack) | onLoanTo(DLHandBook,Jack) |

```
\(\forall x(\exists y\).OnLoanTo \((x, y) \rightarrow \operatorname{LibItem}(x))\)
\(\forall x(\exists y\). OnLoanTo \((y, x) \rightarrow \operatorname{Member}(x))\)
\(\forall x(\operatorname{Member}(x) \rightarrow \operatorname{Person}(x))\)
\(\forall x(\operatorname{Visitor}(x) \rightarrow \operatorname{Person}(x))\)
\(\forall x(\operatorname{Visitor}(x) \rightarrow \neg \operatorname{Member}(x))\)
LibItem(S W Printer)
onLoanTo(DLHandBook,Jack)
```

In this example, the concept LibItem denotes library items, while the role onLoanTo denotes the loan relationship between library items and people. Note that the existential quantification constructor ( $\exists$ ) used in the first axiom allows one to describe the domain of the role. Conversely, in the second axiom, it is the use of the inverse role constructor $\left(^{-}\right)$, in addition to the $\exists$ constructor, that allows for the specification of the role range. Finally, not every Person is a Member or a Visitor, and a Visitor is not a Member.

Given the need to modify, reuse, decompose and combine different ontologies, the study of the forgetting in DLs has been gaining increasing preponderance, as these proceedings typically require omission of terms or attention restriction to a particular set of concepts, roles and individuals while preserving all the logical consequences between the remaining [Wan+08]. In fact, these are regarded as quite complex tasks considering that concepts in ontologies (as well as their corresponding relationships) are generally highly coupled. This requires sophisticated methods and operations to deal with the problem. To this end, several approaches to forgetting concept or role symbols either from $\mathcal{T}$ or from $K B=\langle\mathcal{T}, \mathcal{A}\rangle$ obeying useful properties have been widely discussed [EK19].

For instance, Wang et al. [Wan+08] focused on forgetting concepts in the DL-Lite family of description logics. The authors start by providing a semantic (i.e., modelbased) definition of forgetting from TBoxes. Intuitively, a TBox $\mathcal{T}^{\prime}$ that results from forgetting about a concept $A$ in a TBox $\mathcal{T}$ should not contain any occurrence of $A$, be weaker than $\mathcal{T}$ and give the same answer to any query for which $A$ is irrelevant. This idea is captured through a semantic characterisation supported by the notion of similarity between models, generalising the approach for classical logic (see Sections 2.1 and 2.2). More specifically, the concept $A$ must not be on the signature of $\mathcal{T}^{\prime}$, and the models of $\mathcal{T}$ and $\mathcal{T}^{\prime}$ must agree on everything except possibly $A$. This definition implies that the result of forgetting about $A$ in $\mathcal{T}$ is unique. In addition, similarly to what we saw earlier for predicate logic and FOL, the definition can be extended to a set of concepts by applying an iterative process, where the order is irrelevant to the final result.

Further in their work, Wang et al. show that $\mathcal{T}^{\prime}$ can always be obtained from $\mathcal{T}$ through syntax-based transformations. For that, the authors introduce a set of algorithms that compute the result of forgetting for different languages of DL-Lite. These algorithms have important properties: they run in polynomial time and are sound and complete,
guaranteeing that the result of forgetting about concepts always exists. In specific, if we wished to forget a concept $A$ from a DL-Lite ${ }_{\text {core }}$ TBox $\mathcal{T}$, first we would (1) remove axiom $A \sqsubseteq A$ from $\mathcal{T}$ if it is present; then (2) if axiom $A \sqsubseteq \neg A$ is in $\mathcal{T}$, we remove each axiom $A \sqsubseteq C$ or $B \sqsubseteq \neg A$ from $\mathcal{T}$, and replace each axiom $B \sqsubseteq A$ in $\mathcal{T}$ by $B \sqsubseteq \neg B$ (here $C$ is a concept expression and $B$ is a concept expression not negated ( $\neg)$ ); (3) replace each axiom $B \sqsubseteq \neg A$ in $\mathcal{T}$ by $A \sqsubseteq \neg B ;(4)$ for each exiom $B_{i} \sqsubseteq A(1 \leq i \leq m)$ in $\mathcal{T}$ and each axiom $A \sqsubseteq C_{j}$ ( $1 \leq j \leq n$ ) in $\mathcal{T}$, we add the axiom $B_{i} \sqsubseteq C_{j}$ to $\mathcal{T}$ if it is not already there; and finally (5) remove every axiom containing $A$ in $\mathcal{T}$.

Example 2.7 (cf. [Wan+08]). Consider the TBox $\mathcal{T}$ in Example 2.6. If we wish to forget about the concept Member in $\mathcal{T}$ because, e.g., the library now wants to allow nonmembers to borrow items but still wishes to prevent visitors from doing so, then by applying the algorithm steps: (1) and (2) are skipped since the axioms Member $\sqsubseteq$ Member and Member $\sqsubseteq \neg$ Member are not in $\mathcal{T}$; (3) the axiom Visitor $\sqsubseteq \neg$ Member is replaced by Member $\sqsubseteq \neg$ Visitor; (4) the axioms $\exists$ OnLoanTo ${ }^{-} \sqsubseteq$ Person and $\exists$ OnLoanTo ${ }^{-} \sqsubseteq \neg$ Visitor are added to $\mathcal{T}$; (5) the axioms $\exists$ OnLoanTo ${ }^{-} \sqsubseteq$ Member, Member $\sqsubseteq$ Person and Member $\sqsubseteq$ $\neg$ Visitor are removed from $\mathcal{T}$; we would obtain the TBox $\mathcal{T}^{\prime}$ with the axioms
$\exists$ OnLoanTo $\sqsubseteq$ LibItem, ヨOnLoanTo- $\sqsubseteq$ Person,
$\exists$ OnLoanTo $^{-} \sqsubseteq \neg$ Visitor, Visitor $\sqsubseteq$ Person.

As for the definition of forgetting about concepts in DL-Lite ABoxes, it is not always possible to express syntactically in the language of the ABox the semantic result [Wan+08]: consider, for instance, a DL-Lite ${ }_{\text {core }}$ knowledge base $K B$ with a TBox $\mathcal{T}=\{$ Visitor $\sqsubseteq$ $\neg$ Member $\}$ and an ABox $\mathcal{A}=\{V$ isitor(anna) \}. To forget about the concept Visitor in $\mathcal{A}$, we have to remove the assertion $\operatorname{Visitor(anna).~However,~from~} \mathcal{T}$ we can conclude $\neg \operatorname{Member}($ anna), but by definition this cannot be expressed in the language of the ABox. This implies that the information about anna not being a member would also be lost when forgetting about the concept Visitor.

Generally, this happens to other DLs as well, not only for ABoxes but also for TBoxes, which makes forgetting in knowledge bases a difficult challenge (in fact, for some languages, the existence of forgetting a concept is undecidable). Nonetheless, there are some ways of overcoming this issue. For instance, one can extend the language, or even introduce extra vocabulary [EK19].

Having said that, the study of forgetting in DLs remains largely an open problem. For further details on the landscape, the interested reader can resort to [EK19].

### 2.4 Forgetting in Answer Set Programming

We now focus on forgetting in non-monotonic logic and specifically on Answer Set Programming, a logic programming formalism designed to easily represent and efficiently solve complex combinatorial problems in different domains [SW18]. As such, given the
richness of ASP constructors for declarative knowledge representation allied to the existence of highly efficient solvers for program evaluation, this formalism has recently been gaining increasing popularity both in academia and industry [EK19].

The growing interest in ASP has lead to the study of several reasoning operations within the paradigm, some of which already discussed in other formalisms. However, when it comes to forgetting in ASP, similarly to what happens with DLs, this operation presents complex challenges, albeit due to distinct reasons. First, some of these challenges can be exactly attributed to the non-monotonic nature of ASP where, contrary to a monotonic formalism (e.g., any of the logics discussed previously), learning a new fraction of knowledge can mean a reduction on the set of what is known. In that regard, assume as an example (c.f. [EK19]) the following ASP program

$$
K B=\{\operatorname{student}(\operatorname{sam}) . \quad \text { single }(x) \leftarrow \operatorname{student}(x), \text { not married }(x) .\},
$$

where the first rule represents the fact "sam is a student" and the second rule means "if a student is not known to be married, then s/he must be single". Note that the default negation (not) in the second rule allows to express incomplete information, which consequently enables us to conclude that by default (i.e., lack of evidence on the contrary) sam must be single. Therefore, this program has a unique answer set (i.e., a solution ${ }^{4}$ ) $M=\{$ student(sam), single(sam) $\}$. However, if we add the fact married(sam) to program $K B$ (e.g., because we learn that sam is in fact married), we can no longer conclude that sam is single, and thus the answer set for the augmented program would be $M^{\prime}=\{$ student(sam), married (sam) $\}$.

Because of this behaviour, non-monotonic formalisms are viewed as adequate representations of common sense reasoning since, in the real world, humans are frequently required to deal with uncertain and incomplete information and often change their convictions about the world. Note that this is not possible to represent with monotonic formalisms (e.g., classical logic) since any extension to a knowledge base cannot change what was originally inferred [BL04]. Nevertheless, with the convenience of ASP as a way of emulating human reasoning due to its non-monotonic characteristics also comes the difficulty of defining appropriate notions of forgetting in the language.

Furthermore, one other reason that poses said difficulties is the fact that the syntax of the rules that constitute any program matter in such a way that, even if two programs $K B$ and $K B^{\prime}$ have the same answer sets, the result of extending both with the same set of rules may lead to different answer sets for the augmented programs [LPV01]. Eiter and Kern-Isberner [EK19] give an intuitive example that shows exactly this. Suppose that $K B=\{p u b \leftarrow$ thirsty, not sunday. $\}$ and $K B^{\prime}=\{p u b \leftarrow$ thirsty, weekend. $\}$. In this case, both $K B$ and $K B^{\prime}$ have the same answer set $M=\emptyset$. However, if we add the fact thirsty to both programs, then the answer set for the augmented $K B$ would be $\{p u b\}$

[^7]while the answer set for $K B^{\prime}$ would not change. On the contrary, when two programs remain equivalent after any extension, we say that they are strongly equivalent ${ }^{5}$.

In view of these challenges, it is still difficult to reach a common ground when it comes to forgetting atoms in answer set programs, and contrarily to what happens in classical logic, it cannot be pinpointed to a unique definition. In fact, the study of forgetting in ASP is still an active field of research, where several operators (or classes of operators ${ }^{6}$ ) and desirable properties remain under discussion [EK19]. Among the many operators, some focus on syntactic rule transformations, while others view forgetting as a semantic operation [GKL16a; Lei17]. Moreover, in what concerns the proposed properties, these often motivate new operators and approaches to forgetting. Notably, some properties share intricate relationships with others like, e.g., incompatibility, implication or even equivalence [GKL16a]. Regardless, among these properties, strong persistence [KA14], which is closely related to strong equivalence of programs, is particularly interesting, as it seems to best encode what is expected from forgetting in ASP [Lei17]. Intuitively, strong persistence ensures that the result of forgetting an atom from a program does not affect the existing relations between the atoms not to be forgotten. More formally, an operator forget obeys strong persistence if for any program $K B$, atom to be forgotten $p$ and program $K B^{\prime}$ not involving $p$, it holds that

$$
\operatorname{Mod}_{\Sigma \backslash\{p\rangle}\left(K B \cup K B^{\prime}\right)=\operatorname{Mod}\left(f \operatorname{orget}(K B, p) \cup K B^{\prime}\right),
$$

where $\operatorname{forget}(K B, p)$ denotes the result of forgetting $p$ from $K B$, and $\operatorname{Mod}_{\Sigma}(K B)$ denotes the answers sets of $K B$ over the signature $\Sigma$. Although desirable, this property is not always possible to guarantee when forgetting a set of atoms from a program, as those atoms may play such a decisive role in the program that one cannot simply forget them and still expect that the relations between the remaining stay intact [GKL16b; Gon+20]. The authors showed exactly this with the following example. Consider the knowledge base

$$
K B=\{a \leftarrow p . \quad b \leftarrow q . \quad p \leftarrow \operatorname{not} q . \quad q \leftarrow \operatorname{not} p .\} .
$$

It is not possible to forget both atoms $p$ and $q$ from $K B$ and, at the same time, preserve the original semantic relations between $a$ and $b$, since no program over atoms $\{a, b\}$ would have the same answer sets of the original program (over signature $\Sigma \backslash\{p, q\}$ ) when both are extended by a third program over atoms $\{a, b\}$. In those cases, one can resort to operators specifically defined for when strong persistence is not achievable [Gon+17; Gon +20 ]. The choice of ideal operator, however, turns out to be highly dependent on the application domain.

At last, given the complex panorama of forgetting in ASP, the work [GKL16a] makes extensive efforts towards unifying it. For that, the authors examine in detail the literature,

[^8]which includes existing operators, properties and algorithms/implementations, and show results that go well beyond it.

### 2.5 GDPR and Data Encryption

With the recent GDPR and the ever-growing concerns about data privacy, not only in the EU but also around the world, it is not surprising that several systems, tools and procedures have been proposed as a way to protect users' personal data (e.g., by reducing or preventing unnecessary processing or by imposing data confidentiality) while still ensuring as much system functionality as possible [VBO03]. Among them, we can name encryption techniques, which will be the object of reflection in this section given their recurrent discussion in the literature and the existence of several widely known system implementations. In particular, we focus on how relational data encryption can be seen in light of the aforesaid Regulation and how it differs from our interpretation of a desirable forgetting operator for relational databases.

In that respect, according to the GDPR, applying encryption techniques to user data in such a way that it can no longer be attributed to a specific individual is regarded as pseudonymisation ${ }^{7}$. In practice, companies are highly advised to implement these technical measures in the conception of their systems and applications ${ }^{8}$. Still, pseudonymising users' data in order to protect their privacy may convey a false sense of security both for the organisation and the data subject, since the process can be easily nullified with access to additional information (for instance held by a third party, or posteriorly collected by the organisation in question), or even reverted with the corresponding decryption key. For these reasons, pseudonymisation does not conform with the right to erasure (or 'right to be forgotten').

Alternatively, data anonymisation can be a way of circumventing said right as long as adequate mechanisms are applied ${ }^{9}$. In practice, anonymisation can be achieved by means of encryption when the key used to decrypt the data is deleted (or, to the same effect, given to the data subject). Furthermore, by ensuring that the right data is irreversibly altered, in such a way that the link with the user it describes is completely erased, organisations guarantee full compliance with the GDPR. As a matter of fact, when the data can no longer be associated with its corresponding data subject (i.e., it is effectively anonymous), then it simply ceases to belong to the user ${ }^{10}$.

Therefore, and without delving into further details regarding the Regulation nor the technical specifications of these methods, we can attribute to data encryption an important role when it comes to de-identify a data subject.

[^9]In the context of relational databases, this is done by encrypting specific values in a database table (in particular, personally identifiable information such as citizen card number, social security number, full name, or any other natural identifiers, as well as attributes such as age, gender, ethnicity, job title, postal code, etc.). What is more, other sensitive information that at first may not explicitly identify a particular user can still be prompt to inference attacks and thus should receive the same treatment (e.g., medical conditions and interventions, education, religion, among others) [NS10]. By encrypting some of the data instead of deleting it, it is possible to avoid null values and, in most cases, not completely impair data processing for marketing (or other) purposes.

Following these lines, systems such as CryptDB [Pop+11] and Cipherbase [Ara+13] have shown the feasibility of these practices, protecting confidential data from malicious actors or curious administrators while supporting query answering and other critical database system functionalities over the encrypted data. Moreover, these systems allow selection among different degrees of encryption, which enables organisations to handpick the attributes they deem relevant to encrypt at each level. Interestingly, CryptDB is also able to chain encryption keys to user passwords so that only users can decrypt their data.

However, whereas the latter feature is in line with what is expected from the process of anonymisation as far as the GDPR is concerned, the former falls short on the definition since the data is not truly irreversibly altered (i.e., the decryption key destroyed or given to the user). Additionally, depending on the level of encryption chosen, it may still be possible to derive insights on some attributes, which can eventually result in the reidentification of an individual. As a result, encryption alone is generally no silver bullet.

Thus, to overcome these issues, in some cases, other techniques for anonymisation are possible on top of the data encryption process. These, in fact, may be even more convenient and in compliance with the GDPR. For instance, some user attribute values can be generalised instead of encrypted (e.g., date of birth can be converted to year of birth, and heights of individuals changed to more generic categories). This brings additional advantages in the sense that more data may be considered for analysis [Eyu+18].

Nonetheless, after applying these methods, organisations still have to guarantee with absolute certainties that the data cannot be traced in any way back to the user, which usually is not an easy task to do. As we saw, attributes that at first may not be considered personally identifiable, when combined, can still give unwanted cues on who its subject is [NS10]. In fact, to guarantee the level of confidentiality expected to conform with the right to be forgotten, a large set of attributes have to be either irreversibly anonymised or transformed into very broad categories, implying that less and less information is actually valuable. Consequently, this leads to useless storage space in a database, which only affects the scalability of applications and hinders processing and analytical times.

Additionally, along the same lines of what was discussed at the end of Section 2.2, in general, the process of anonymisation makes clear allusions to what operations were done in the past. This, in turn, might not be always desirable, or even legal. For example, what if an organisation has to forget a specific table in a database due to judicial reasons
(e.g., because said information was unlawfully collected or allows for complete disclosure of a large set of individuals)? In that case, simply anonymizing the data is not possible. Perhaps, at first, a viable solution would be erasing all the data and delete the database table. This action, on the other hand, might damage on cascade the processing of other legitimately collected information that may serve important purposes and should not be affected. For example, data used for reasons of public interest, such as public health as well as scientific and statistical research purposes should be held irrespectively of the right to be forgotten ${ }^{11}$.

Given these reasons, a forgetting operator for relational databases in the vein as what is proposed in the literature for other knowledge representation formalisms is of high interest, not necessarily as a replacement for data encryption but rather as an auxiliary tool. This way, it is possible to remove (read forget) information in such a manner that indirect relationships between other data are minimally affected, as we will see with a practical example in Section 4.1.

### 2.6 Schema Evolution

Database systems are responsible for collecting and managing large amounts of information. Given the complexity of this task, it is essential to start the development of a database using appropriate design and modeling techniques. Only this way it is possible to faithfully represent in a database the different concepts found in a specific knowledge domain. When it comes to the design process, one important step is to outline the database schema, which describes how the data is structured in the database.

Even though it does not happen very frequently [SKS11], most database schemas undergo several changes during their lifetime, either because there is an explicit need to accommodate new application requirements, due to performance reasons, or simply to fix previous mistakes [ALP91; Rod92]. These changes include inter alia adding, removing or renaming database tables, adding new attributes to tables, or even renaming and deleting existing ones. As such, it is highly desirable that database systems provide the necessary manipulation tools to address said needs.

In that sense, schema evolution (or schema change) refers to the ability of a database system to deal with modifications imposed to deployed schemas [RB06]. However, support for schema evolution is a challenging task since it presupposes accurate and efficient propagation of schema changes to the data, queries, views as well as other dependent schemas and applications [HTR11].

Considering that schema evolution during the operational phase is an intricate problem that, when incorrectly employed, might lead to serious consequences regarding the integrity of the data [ALP91; CMZ09; Sjø93], several techniques, systems and tools have been proposed in the literature to assist database systems support this operation [ALP91;

[^10]HTR11; RB06; Rod92]. One such example is to be found in the work of Curino et al. [CMZ09]. The authors present a system to automate the database schema evolution process, which is capable of rewriting queries and updates. That way, the authors are able to safeguard data integrity and minimise system downtimes while supporting schema upgrades.

As we will see in the next chapters, forgetting in relational databases (e.g., to comply with the GDPR's 'right to be forgotten') may imply schema changes as well. In those cases, a theoretical formalisation of forgetting operators as proposed in this dissertation, will help organisations understand which changes should be applied to their databases at the level of the schema and subsequently the data and queries. Only then can organisations leverage systems supporting schema evolution capabilities to facilitate the implementation of the defined changes while minimally affecting their daily operations.

## A Model for Relational Databases

The relational data model was proposed by E. F. Codd in 1970 [Cod70]. This model provides a simple and intuitive yet powerful way to represent information. For these reasons, it is nowadays widely used as the principal data model for data-processing applications [SKS11].

Still, in order to facilitate the development of our theory of forgetting, we require a model for relational databases that is more closely aligned with our needs. Therefore, in this chapter, we revisit the relational model and introduce a new formalisation that is specifically tailored to deal with the problem of forgetting in relational databases. Ultimately, this means that in some cases we simplify some of the concepts that are known in the database literature, while in others we extend them. Nevertheless, whenever convenient, we merely adapt them to our notation and terminology.

More concretely, in the first sections, we present the building blocks that constitute the relational model, namely the concept of a relation, an integrity constraint (focusing especially on functional dependencies), a relational database and a database query. Finally, in the last section, we introduce a novel definition of equivalence between relational databases, which addresses some of the shortcomings found for similar notions in the literature.

### 3.1 Relation

In the relational model, information is represented in relations. A relation can be viewed as a table, where a row represents a tuple, and a column represents an attribute. Therefore, we can use the terms tuple and attribute to refer to a row, respectively a column, in a table. For instance, the table in Figure 3.1 is a visual representation of the relation customer, which stores information about the customers of a particular service. This relation has the attributes ID, Name and YearOf Birth, corresponding to the columns of the table, and one of its tuples is (1, Anna, 1990), which corresponds to the first row. Each row describes the relationship among the values for the respective attributes. E.g., regarding the previous tuple, the customer with ID 1 is named Anna and was born in 1990.

| customer |  |  |
| :---: | :---: | :---: |
| ID | Name | YearOfBirth |
| 1 | Anna | 1990 |
| 2 | David | 1971 |
| 3 | John | 1990 |
| 4 | Mary | 1987 |

Figure 3.1: The customer relation.

A relation consists of a name, a schema and an instance. In what follows we introduce each concept individually. To that end, let us start by defining the domain of an attribute, which corresponds to the set of permitted values the attribute can take.

Definition 3.1 (Attribute Domain). The domain of an attribute $A$, denoted by $\operatorname{dom}(A)$, is a non-empty countable set of atomic elements.

Going back to our relation customer in Figure 3.1, the domain of the attributes $I D$ and YearOf Birth is the set of all positive integers, while the domain of the attribute Name is the set of all possible strings.

Moreover, from here on, we will assume a fixed signature, that is, a finite set $\mathcal{A}$ of attributes. Further, we also assume a total fixed order for the attributes in $\mathcal{A}$.

Let us now define the schema of a relation.

Definition 3.2 (Relation Schema). The schema of a relation is a finite and ordered set of attributes $\left(A_{1}, A_{2}, \ldots, A_{n}\right)$, where each attribute $A_{i}$, with $1 \leq i \leq n$, is in $\mathcal{A}$. It corresponds to the logical design of the relation.

We draw attention to two details regarding this definition. First, it does not allow for multiple occurrences of the same attribute in a relation schema. Secondly, each attribute $A_{i}$ in a schema is assigned its own domain $\operatorname{dom}\left(A_{i}\right)$.

Furthermore, we define the arity of a relation as the number of attributes in its schema. For example, the schema of the customer relation in Figure 3.1 is (ID,Name, YearOf Birth) and therefore its arity is three. This implies that each tuple in the relation must have three values. In any case, these values are restricted by the respective attribute domains.

In order to talk about the set of tuples that represents the contents of a relation at a given instant in time, we use the term relation instance.

Definition 3.3 (Relation Instance). The instance of a relation is a finite subset of the Cartesian product of its attribute domains, i.e., $\operatorname{dom}\left(A_{1}\right) \times \operatorname{dom}\left(A_{2}\right) \times \ldots \times \operatorname{dom}\left(A_{n}\right)$, where $\left(A_{1}, A_{2}, \ldots, A_{n}\right)$ corresponds to the schema of the relation.

In this dissertation, we do not consider instances with null values.
As usual, if we want to mention the value of a specific attribute $A$ for a tuple $t$, we use the notation $t[A]$. In addition, if $A=\left(A_{1}, A_{2}, \ldots, A_{n}\right)$ is an ordered set of attributes, then
$t[A]$ denotes $\left(t\left[A_{1}\right], t\left[A_{2}\right], \ldots, t\left[A_{n}\right]\right)$. For instance, if $t$ corresponds to the first tuple in the relation shown in Figure 3.1, then $t[I D]=1, t[\mathrm{Name}]=$ Anna and, for $A=(I D, N a m e)$, we have $t[A]=(1$, Anna $)$.

We can now formally define a relation.

Definition 3.4 (Relation). A relation $r$ is a triple $(n(r), s(r), i(r)$ ), where $n(r)$ stands for the relation name, $s(r)$ denotes the schema of the relation and $i(r)$ its instance ${ }^{1}$.

We denote by $\mathcal{R}_{\mathcal{A}}$ be the set of all relations over $\mathcal{A}$, i.e., the set of relations whose schema is contained in $\mathcal{A}$.

In general, the order in which the tuples appear in a relation instance is irrelevant, since it corresponds to a set. However, sometimes we will want to impose an artificial order upon the instances of the relations and, for that, we need to be able to uniquely identify each tuple. To this end, we will assume that the schema of every relation has an extra attribute RowId, whose domain is the positive integers. That being said, we will leave the RowId always hidden and, unless otherwise stated, will not consider nor represent it in any operation with relations.

In the following we introduce the concepts of inclusion and equivalence between relations.

Definition 3.5 (Inclusion between Relations). Let $r, r^{\prime} \in \mathcal{R}_{\mathcal{A}}$. We say that $r$ is included in $r^{\prime}$, denoted by $r \sqsubseteq r^{\prime}$, if they have the same schema and each tuple in the instance of $r$ is also in the instance of $r^{\prime}$, i.e., $s(r)=s\left(r^{\prime}\right)$ and $i(r) \subseteq i\left(r^{\prime}\right)$.

Definition 3.6 (Equivalence between Relations). Let $r, r^{\prime} \in \mathcal{R}_{\mathcal{A}}$. We say that $r$ is equivalent to $r^{\prime}$, denoted by $r \equiv r^{\prime}$, if $r \sqsubseteq r^{\prime}$ and $r^{\prime} \sqsubseteq r$.

Example 3.1. Consider the relation employee represented by the following table and the relation customer depicted in Figure 3.1.

| employee |  |  |
| :---: | :---: | :---: |
| ID | Name | YearOfBirth |
| 1 | Anna | 1990 |
| 3 | John | 1990 |

Since $s($ employee $)=s($ customer $)$ and $i($ employee $) \subseteq i($ customer $)$, then employee $\sqsubseteq$ customer. On the other hand, $i$ (customer) $\nsubseteq i$ (employee). Therefore, employee is included in customer but the relations are not equivalent.

[^11]
### 3.2 Functional Dependency

In order to correctly model the world under consideration, it is often necessary that relation instances conform with a set of rules and properties. For instance, we might want to impose that customers, which are distinguished by their $I D$, cannot have more than one name and year of birth. These rules are declared in the form of integrity constraints. Particularly, in this work, we focus on an expressive type of constraints named functional dependencies (FDs for short). FDs are represented by formulas $X \rightarrow Y$, where $X$ and $Y$ are sets of attributes in $\mathcal{A}^{2}$. Given a relation $r$ and a functional dependency $X \rightarrow Y$ with $X$ and $Y$ being in the schema of $r$, we say that $r$ satisfies $X \rightarrow Y$ when, for all tuples in the instance of $r$, the combination of values in $X$ uniquely determines the combination of values in $Y$. In that case, we say that $X$ functionally determines $Y$ in $r$. We define this notion more formally below.

Definition 3.7 (Functional Dependency Satisfaction). Let $r$ be a relation over $\mathcal{A}$ and $X, Y$ two sets of attributes in the schema of $r$. We say that $r$ satisfies the functional dependency $X \rightarrow Y$, if for all pairs of tuples $t_{1}, t_{2} \in i(r)$ we have that $t_{1}[X]=t_{2}[X]$ implies $t_{1}[Y]=t_{2}[Y]$.

Example 3.2. Consider the relation customer in Figure 3.1. It is easy to see that it satisfies the functional dependency $I D \rightarrow$ Name, since each value in ID occurs only once in the instance of the relation. On the contrary, the functional dependency YearOf Birth $\rightarrow$ Name is not be satisfied by customer, given that for $t_{1}=\left(1\right.$, Anna, 1990) and $t_{2}=(3$, John, 1990 $)$, we have $t_{1}[$ YearOf Birth $]=t_{2}[$ YearOf Birth $]=1990$ but not $t_{1}[\mathrm{Name}]=t_{2}[\mathrm{Name}]$.

We denote by $\mathcal{F}_{\mathcal{A}}$ the set of all functional dependencies over $\mathcal{A}$.
In the next definition we present the notion of closure of a set of FDs, which is widely known in the literature.

Definition 3.8 (Closure of a set of Functional Dependencies). Let $F \subseteq \mathcal{F}_{\mathcal{A}}$, the closure of $F$, denoted by $F^{+}$, corresponds to the set of all FDs in $\mathcal{F}_{\mathcal{A}}$ that must be satisfied by any relation that satisfies the FDs in $F$. We say that those FDs are logically implied by $F$.

Usually, this set can be inferred by applying repeatedly a set of rules, the well-known Armstrong's axioms [Arm74], which are sound and complete [Mai83]. In particular, in this dissertation, we will often take advantage of the transitivity rule as defined below.

Definition 3.9 (Transitivity Rule). Let $X, Y$ and $Z$ be sets of FDs in $\mathcal{A}$. If $X \rightarrow Y$ holds and $Y \rightarrow Z$ holds, then $X \rightarrow Z$ also holds.

What is more, the notion of closure of a set of FDs allows us to establish an idea of equivalence between sets of $F D$ s, which can be defined as follows.

Definition 3.10 (Equivalence between FD Sets). Let $F, F^{\prime} \subseteq \mathcal{F}_{\mathcal{A}}$. We say that $F$ and $F^{\prime}$ are equivalent, denoted by $F \equiv F^{\prime}$, if they imply the same set of FDs , i.e., if $F^{+}=F^{\prime+}$.

[^12]We now introduce some other concepts that will be useful to manipulate sets of FDs.
Definition 3.11 (Projection of a Set of Functional Dependencies in a Relation Schema). Given a relation $r \in \mathcal{R}_{\mathcal{A}}$ and a set of functional dependencies $F \subseteq \mathcal{F}_{\mathcal{A}}$, the projection of $F$ on $s(r)$, which we denote $F_{r}$, is the set of all functional dependencies in $F^{+}$that only include attributes of $s(r)$.

This notion allows us to talk about the FDs that are relevant for a specific relation. However, given a set of relations, sometimes we may want to exclusively mention the FDs that are only relevant for a particular relation. In other words, we may want to talk about those that are projected on its schema, but not on the schema of any other relation in the set. Such intuition is captured by the following definition.

Definition 3.12 (Exclusive Projection of a Set of Functional Dependencies in a Relation Schema). Given a relation $r \in \mathcal{R}_{\mathcal{A}}$, a set of relations $R \subseteq \mathcal{R}_{\mathcal{A}}$ and a set of functional dependencies $F \subseteq \mathcal{F}_{\mathcal{A}}$, the set of FDs in $F$ that are projected on the schema of $r$, but not on the schema of some relation in $R \backslash\{r\}=\left\{r_{1}, \ldots, r_{n}\right\}$, is denoted by $F_{r}^{R}$, and equal to $F_{r} \backslash\left(F_{r_{1}} \cup \ldots \cup F_{r_{n}}\right)$.

A procedure to compute this result is given by Algorithm 1. It starts by creating a new variable $F^{\prime}$ and initialising it as the empty set. This variable will store all the FDs in $F^{+}$(computed in line 2) that are projected exclusively on the schema of $r$. To that end, the algorithm iterates over all FDs in $F^{+}$whose attributes in the right and left-hand side belong to $s(r)$ (lines 3 and 4), and verifies if they also belong to the schema of any other relation in $R$. If that is not the case, then it adds the respective FD to $F^{\prime}$ (lines 10 and 11).

```
Algorithm 1: Computation of \(F_{r}^{R}\)
input :Triple ( \(F, R, r\) ) such that \(F \subseteq \mathcal{F}_{\mathcal{A}}, R \subseteq \mathcal{R}_{\mathcal{A}}\) and \(r \in \mathcal{R}_{\mathcal{A}}\)
output: Set of FDs resulting from compute- \(F_{r}^{R}(F, R, r)\)
    \(F^{\prime} \leftarrow \emptyset ;\)
    compute the closure of \(F\), denoted \(F^{+}\), using procedure in [SKS11];
    foreach \(X \rightarrow Y \in F^{+}\)do
        if \(X \cup Y \subseteq s(r)\) then
            projected \(\leftarrow\) false;
            /* check if all attributes in \(X \rightarrow Y\) belong to the schema of a relation in
                \(R \backslash\{r\}\). */
            foreach \(r^{\prime} \in R \backslash\{r\}\) do
                if \(X \cup Y \subseteq s\left(r^{\prime}\right)\) then
                projected \(\leftarrow\) true;
                    break;
            if not projected then
                    \(F^{\prime} \leftarrow F^{\prime} \cup\{X \rightarrow Y ; ;\)
    return \(F^{\prime}\);
```


### 3.3 Database

A relational database, or database for short, is a collection of relations and FDs, such that the relations satisfy the closure of the FDs. In this context, we assume that all relations in a database have a unique name.

Definition 3.13 (Relational Database). A relational database $D$ over $\mathcal{A}$ is a pair $(R, F)$, where $R$ is an ordered set of relations over $\mathcal{A}$ and $F$ a set of FDs over $\mathcal{A}$, such that the relations in $R$ satisfy the FDs in $F^{+}$.

Notice that expressing FDs in a database in this way differs from what is usual in the literature, where FDs are specified for each relation individually.

Furthermore, as means to refer to the set of relations and the set of FDs of a database $D=(R, F)$, we denote $R_{D}=R$ and $F_{D}=F$. Also, we denote by $\mathcal{D}_{\mathcal{A}}$ the set of all databases over $\mathcal{A}$.

Considering that a database describes the information of a particular world, it is only natural that the data stored in its relations are related to each other. Therefore, for any database $D \in \mathcal{D}_{\mathcal{A}}$, we adopt the unique-role assumption [SKS11], which states that each attribute has a unique meaning in the database. Put differently, the same attributes in different schemas of the relations in $R_{D}$ have always the same meaning (and consequently the same domain).

Definition 3.14 (Schema of the Relations in a Database). Given a database $D=\left(\left(r_{1}, \ldots, r_{n}\right), F\right) \in$ $\mathcal{D}_{\mathcal{A}}$, the schema of the relations in $D$, denoted by $s\left(R_{D}\right)$, is the tuple $s\left(R_{D}\right)=\left(s\left(r_{1}\right), \ldots, s\left(r_{n}\right)\right)$.

The schema of the relations in a database describes how the data is structured in the database. Intuitively, it is the tuple obtained by applying $s(r)$ to all relations $r \in R_{D}$ with respect to the order of $R_{D}$.

Additionally, we will often want to talk about the schema of the database, which is a different concept from the schema of the relations in a database, as it also considers the FDs that belong to the database.

Definition 3.15 (Database Schema). Given a database $D \in \mathcal{D}_{\mathcal{A}}$, the database schema (abbreviated dbs) of $D$, denoted by $\mathcal{S}(D)$, is the pair ( $\Sigma, F)$, such that $\Sigma=s\left(R_{D}\right)$ is a tuple of relation schemas over $\mathcal{A}$ and $F=F_{D}$ is a set of FDs over $\mathcal{A}$.

For the case above, we say that $D$ is an instance of the $d b s(\Sigma, F)$.
Definition 3.16 (Instance of a Database Schema). Given a database $D \in \mathcal{D}_{\mathcal{A}}$ and a dbs $S=(\Sigma, F)$ over $\mathcal{A}$, we say that $D$ is an instance of $S$ if $s\left(R_{D}\right)=\sum$ and $F_{D}=F$.

Finally, we denote by $\mathcal{D}(S)$ the set of all instances of $S$. Or, in other words, the set of all databases whose schema is $S$.

### 3.4 Database Query

To retrieve information from a relational database (i.e., query a database) we resort to relational query languages (or query languages for short). These languages define a set of operations that operate upon the relations of a database with a given schema, and output a relation with a fixed schema. The operations can then be combined to construct expressions that represent desired queries [SKS11]. In essence, a query can also be viewed as a function. To define it in that way, we first introduce some notation.

Let $\sigma$ be a relation schema over $\mathcal{A}$, we denote by $\mathcal{R}(\sigma)$ the set of relations in $\mathcal{R}_{\mathcal{A}}$ whose schema is $\sigma$, i.e., $\mathcal{R}(\sigma)=\left\{r \in \mathcal{R}_{\mathcal{A}} \mid s(r)=\sigma\right\}$.

We are now ready to define a query as a function.
Definition 3.17 (Query). Let $S$ be a dbs over $\mathcal{A}$. A query on $S$ is a function $q: \mathcal{D}(S) \rightarrow \mathcal{R}(\sigma)$, where $\sigma \subseteq \mathcal{A}$, which takes as input a database with schema $S$ and outputs a relation with schema $\sigma$. Given $D \in \mathcal{D}(S)$, we call $q(D)$ the answer to the query $q$.

One such query language, which will be used frequently in this document, is the relational algebra. We consider the relational algebra operators as they were defined in [SKS11], namely the operators of selection ( $\sigma$ ), projection ( $\Pi$ ), set difference $(-)$, Cartesian product $(\times)$ and natural join $(\bowtie)$. Furthermore, we will also consider a particular sublanguage of the relational algebra, which can be obtained by restricting the queries in this language to the conjunctive queries. These can be defined as follows.

Definition 3.18 (Conjunctive Relational Algebra Query [AIR99]). A conjunctive query in the relational algebra corresponds to any query written exclusively with the operators of selection, projection and natural join, or combinations among them.

### 3.5 Database Equivalence

In this section, we present a novel definition of database equivalence. To do so, we first lay out the concepts of derivability between database schemas as well as derivability between databases. While the former is based on the ability of a database schema to represent the information that can be stored on another schema through a collection of queries, the second compares two specific instances of said schemas, evaluating whether one database can emulate exactly the contents of the other via the same set of queries that guarantees derivability between the respective schemas.

These notions will be used throughout this work, as they are relevant for the study of transformations at the level of the schema of the database and therefore particularly meaningful in the context of forgetting.

In particular, our definition of derivability between database schemas is based on the concept of "weak inclusion between databases" formally proposed by Ausiello et al. [ABM80] and whose roots go back to the work of Codd in [Cod72], namely with the
notions of "derivability and query-equivalence between instances of databases" ${ }^{3}$. In fact, the terms used by Codd motivate the ones employed herein.

Moreover, similarly to what is done by Ausiello et al., we parameterise the concepts to be introduced with a query language $\mathcal{Q}$. Thus, by definition, derivability (and consequently equivalence) between databases will be dependent on the chosen query language. Ultimately, this means that they may hold for more expressive query languages, but not for specific subsets of those languages.

Definition 3.19 (Derivability with respect to $\mathcal{Q}$ between Database Schemas). Let $S=$ $\left(\left(\sigma_{1}, \ldots, \sigma_{n}\right), F\right)$ and $S^{\prime}$ be two dbs over $\mathcal{A}$ and $\mathcal{Q}$ a query language. We say that $S$ is derivable from $S^{\prime}$ with respect to $\mathcal{Q}$ (abbreviated $\mathcal{Q}$-derivable and denoted $S \leqslant_{\mathcal{Q}} S^{\prime}$ ) if there is a tuple $\bar{q}=\left(q_{1}, \ldots, q_{n}\right)$ of queries in $\mathcal{Q}$ such that, for every database $D \in \mathcal{D}(S)$, there exists a database $D^{\prime} \in \mathcal{D}\left(S^{\prime}\right)$ where, for each relation $r_{i} \in R_{D}$, we have $r_{i} \sqsubseteq q_{i}\left(D^{\prime}\right)$, for $1 \leq i \leq n$.

Regarding the definition above, we will often write $S \leqslant_{\mathcal{Q}} S^{\prime}$ by $\bar{q}$ to indicate the tuple of queries that establishes derivability between $S$ and $S^{\prime}$.

Intuitively, a dbs $S$ is $\mathcal{Q}$-derivable from a dbs $S^{\prime}$ if, for every instance of $S$, there is an instance of $S^{\prime}$ from which we can extract at least the same information by means of queries in the language $\mathcal{Q}$. This means that, if a dbs is $\mathcal{Q}$-derivable from another, then the latter can always represent the information that is stored in the former. Notice that, for this to be possible, the relation schemas resulting from the queries on $S^{\prime}$ must match the ones of the relations in $S$.

The following example illustrates $\mathcal{Q}$-derivability between database schemas.
Example 3.3. Assume that $\mathcal{Q}$ corresponds to the conjunctive queries in the relational algebra. Furthermore, consider two database schemas $S=(\Sigma, F)$ and $S^{\prime}=\left(\Sigma^{\prime}, F^{\prime}\right)$ such that

- $\Sigma=((I D, N a m e, Y e a r O f$ Birth $))$,
- $\Sigma^{\prime}=((I D, N a m e),(I D, Y$ earOf Birth $))$, and
- $F=F^{\prime}=\{I D \rightarrow$ Name, YearOf Birth $\}$.

First, note that any instance of $S$ has a single relation $r_{1}$ whose schema corresponds to (ID, Name, YearOf Birth). Likewise, any instance of $S^{\prime}$ has two relations $r_{1}^{\prime}$ and $r_{2}^{\prime}$ such that $s\left(r_{1}^{\prime}\right)=(I D, N a m e)$ and $s\left(r_{2}^{\prime}\right)=(I D$, YearOf Birth $)$. Now, consider the queries

- $q_{1}=r_{1}^{\prime} \bowtie r_{2}^{\prime}$,
- $q_{1}^{\prime}=\Pi_{I D, \text { Name }}\left(r_{1}\right)$, and
- $q_{2}^{\prime}=\Pi_{I D, Y e a r O f B i r t h}\left(r_{1}\right)$.

[^13]Then, it is true that $S \leqslant_{\mathcal{Q}} S^{\prime}$ by $\bar{q}=\left(q_{1}\right)$ and $S^{\prime} \leqslant_{\mathcal{Q}} S$ by $\bar{q}^{\prime}=\left(q_{1}^{\prime}, q_{2}^{\prime}\right)$.
For instance, if $r_{1}$ corresponds to the relation customer from Figure 3.1 (note that it has the same schema as $r_{1}$ ), then $r_{1}^{\prime}$ and $r_{2}^{\prime}$ could be the relations

| $r_{1}^{\prime}$ |  |  | $r_{2}^{\prime}$ |  |
| :---: | :---: | :---: | :---: | :---: |
|  | ID | Name |  | ID |
|  | YearOfBirth |  |  |  |
| 1 | Anna |  | 1 | 1990 |
| 2 | David |  | 2 | 1971 |
| 3 | John |  | 3 | 1990 |
| 4 | Mary |  | 4 | 1987 |

and we would have $r 1 \sqsubseteq r_{1}^{\prime} \bowtie r_{2}^{\prime}$. In fact, for every instance of $r_{1}$, we would always be able to find an instance of $r_{1}^{\prime}$ and $r_{2}^{\prime}$ such that their natural join satisfies this condition. In practice, we just have to project $r_{1}$ on the schema of $r_{1}^{\prime}$ and $r_{2}^{\prime}$.

Regarding derivability in the other direction, if the instances of $r_{1}^{\prime}$ and $r_{2}^{\prime}$ had the same values for the attribute $I D$, then we could find $r_{1}$ by joining both relations. Yet, this strategy would not work if the values for ID did not coincide completely, as the natural join would lead to loss of information. For instance, consider the following tables, which represent relations $r_{1}^{\prime}$ and $r_{2}^{\prime}$.

| $r_{1}^{\prime}$ |  |  | $r_{2}^{\prime}$ |  |
| :---: | :---: | :---: | :---: | :---: |
| ID | Name |  |  |  | | ID | YearOfBirth |
| :--- | :--- | :--- |
| 1 | Anna |
| 4 | Mary |

In this case, a relation $r_{1}$ from which we could extract at least the same information as $r_{1}^{\prime}$ and $r_{2}^{\prime}$ would be the relation represented by the following table.

| $r_{1}$ |  |  |
| :---: | :---: | :---: |
| ID | Name | YearOfBirth |
| 1 | Anna | 1990 |
| 3 | $*$ | 1990 |
| 4 | Mary | $*$ |

To deduce this $r_{1}$, we could simply fill its instance with all the values of $r_{1}^{\prime}$ and $r_{2}^{\prime}$. However, since in these circumstances $r_{1}$ would always have more information than the relations $r_{1}^{\prime}$ and $r_{2}^{\prime}$ combined, and null values are not allowed in our relations, then we would need to replace the * with any values from the domain of the attributes that would respect the FDs. This way, we can obtain at least all the information in $r_{1}^{\prime}$ and $r_{2}^{\prime}$ from $r_{1}$ by using the queries $q_{1}^{\prime}$ and $q_{2}^{\prime}$, respectively.

Regarding the example above, notice that these results are only true because the constraints imposed by the FDs in the instances of both schemas are the same. For
example, if $F^{\prime}$ did not have any FD, then we would have instances of $S^{\prime}$ where the relations had tuples with multiple occurrences of the same $I D$. This, of course, would not be possible for the instances of $S$ due to the FDs. Therefore, $S^{\prime} \leqslant \mathcal{Q} S$ would cease to be true, as it would exist at least one instance of $S^{\prime}$ that we could not represent with an instance of $S$. Anyway, removing the FDs in $F^{\prime}$ would not have any affect in the $\mathcal{Q}$-derivability in the opposing direction, i.e., $S \leqslant_{\mathcal{Q}} S^{\prime}$ because if $S$ is $\mathcal{Q}$-derivable from $S^{\prime}$ then, by deleting some of the constraints in $S^{\prime}$, we will only get more allowed instances and thus $\mathcal{Q}$-derivability will still hold. In general, this example shows the importance of the role that the FDs play in the notion of $\mathcal{Q}$-derivability between database schemas.

Having described $\mathcal{Q}$-derivability between database schemas, we are now ready to extend this definition for databases.

Definition 3.20 (Derivability with respect to $\mathcal{Q}$ between Databases). Let $D=\left(\left(r_{1}, \ldots, r_{n}\right), F\right)$ and $D^{\prime}$ be two databases over $\mathcal{A}$ and $\mathcal{Q}$ a query language. We say that $D$ is derivable from $D^{\prime}$ with respect to $\mathcal{Q}\left(\right.$ abbreviated $\mathcal{Q}$-derivable and denoted $\left.D \leqslant_{\mathcal{Q}} D^{\prime}\right)$ if there exists a tuple $\bar{q}=\left(q_{1}, \ldots, q_{n}\right)$ of queries in $\mathcal{Q}$ such that

- $\mathcal{S}(D) \leqslant_{\mathcal{Q}} \mathcal{S}\left(D^{\prime}\right)$ by $\bar{q} ;$ and
- $r_{i} \equiv q_{i}\left(D^{\prime}\right)$, for each $1 \leq j \leq n$.

The definition above implies that the condition for $\mathcal{Q}$-derivability between databases is satisfied when there is derivability in the same direction between the schemas of those databases and the tuple of queries (within query language $\mathcal{Q}$ ) that guarantees it also allows to retrieve exactly each relation of the first database in the other database. Simply put, there must be a transformation to the relations in the second database thats leads to each relation in the first.

In reality, this last condition alone corresponds to the idea of derivability informally introduced by Codd for the relational algebra [Cod72] and later formalised by Ausiello et al. under the notion of "strong inclusion between database instances" [ABM80], which, to the best of our knowledge, is the only work in the literature that formally defines the concept of derivability (and consequently equivalence) between databases. However, as the following example shows, we argue that these notions are too broad to correctly capture the intuition behind $\mathcal{Q}$-derivability from a database to another, hence our alternative definition.

Example 3.4. Assume that $\mathcal{Q}$ is the conjunctive queries in the relational algebra. Furthermore, consider a database $D$ with a single relation employee, which is represented by the table below.

| employee |  |  |
| :---: | :---: | :---: |
| ID | Name | YearOfBirth |
| 1 | Anna | 1990 |
| 3 | John | 1985 |

Additionally, consider a second database $D^{\prime}$ with one relation employee', obtained by swapping the names and years of birth of the employees Anna and John in the relation above.

| employee' $^{2}$ |  |  |
| :---: | :---: | :---: |
| ID | Name | YearOfBirth |
| 1 | John | 1990 |
| 3 | Anna | 1985 |

Then, for the query

$$
\begin{aligned}
q= & \left.\sigma_{(I D=1 \wedge \text { Name }} \text { "Anna" } \wedge \text { YearOfBirth }=1990\right) \vee(I D=3 \wedge \text { Name="John" } \wedge \text { YearOf Birth }=1985) \\
& \left(\Pi_{I D}(\text { employee }) \bowtie \Pi_{\text {Name }}(\text { employee }) \bowtie \Pi_{\text {YearOfBirth }}(\text { employee })\right)
\end{aligned}
$$

we have employee $\equiv q\left(D^{\prime}\right)$. That is, if we project each attribute of the relation employee', creating three relations with a single attribute, join them (note that this will result in a tuple for every possible combination of the values) and select the tuples of employee, we obtain exactly that relation. In a similar way we could also derive employee' from employee.

A notion of $\mathcal{Q}$-derivability between databases based exclusively on the second condition in Definition 3.20 would imply that the databases $D$ and $D^{\prime}$ from the previous example are $\mathcal{Q}$-derivable from each other, which would not be desirable at all, given the instance of the relation in each database. Thus, by requiring that the tuple of queries that guarantees $\mathcal{Q}$-derivability has to do it for all possible instances of the relations (although for inclusion instead of equivalence), we are adding some "structure" to the notion of $\mathcal{Q}$-derivability between databases that does not exist in the definition of Ausiello et al. This, in turn, avoids that databases such as $D$ and $D^{\prime}$ from the example are derivable from each other (note that the query $q$ would not work for all instances of employee; e.g., it would not work for any instance with a tuple whose ID is different from 1 and 3).

In fact, this example clearly demonstrates the value of imposing $\mathcal{Q}$-derivability between the schemas of the databases in the definition of $\mathcal{Q}$-derivability between databases, considering that it conveniently limits the set of allowed queries that satisfy the second condition of the definition. Moreover, the reason why the definition for database schemas requires inclusion for each relation while the one for databases requires equivalence, has to do with the fact that we do not want to immediately rule out $\mathcal{Q}$-derivability for databases with schemas such as $S$ and $S^{\prime}$ from Example 3.3 (in particular, for the second direction discussed, i.e., $S^{\prime} \leqslant \mathcal{Q} S$ ).

On another subject, note that the proposed definition for $\mathcal{Q}$-derivability between databases presupposes that for every query (written in the language $\mathcal{Q}$ ) on the schema of the derivable database, there is a corresponding query (obtained via a transformation within the same language) on the schema of the database from which it derives that yields exactly the same answer.

We shall call the function that maps each query on the first schema to a query on the second a query mapping.

Proposition 3.1. Let $D$ and $D^{\prime}$ be two databases over $\mathcal{A}$ and $\mathcal{Q}$ a query language. Then, $D \leqslant_{\mathcal{Q}} D^{\prime}$ implies that there is a query mapping $f: \mathcal{Q} \rightarrow \mathcal{Q}$ such that for all queries $q \in \mathcal{Q}$, we have $q(D) \equiv f(q)\left(D^{\prime}\right)$.

Proof. If $D \leqslant_{\mathcal{Q}} D^{\prime}$ holds then there exists a tuple of queries $\bar{q}$ on the schema of $D^{\prime}$ that transforms $D^{\prime}$ into any relation of $D$. Hence, the query mapping $f$ can be obtained by composing $\bar{q}$ with each query $q$.

Whenever two databases are derivable from each other (with respect to a query language), we say that they are equivalent with respect to that language.

Definition 3.21 (Equivalence with respect to $\mathcal{Q}$ between Databases). Let $D$ and $D^{\prime}$ be two databases over $\mathcal{A}$ and $\mathcal{Q}$ a query language. We say that $D$ and $D^{\prime}$ are equivalent with respect to $\mathcal{Q}$ (abbreviated $\mathcal{Q}$-equivalent and denoted $\left.D \equiv_{\mathcal{Q}} D^{\prime}\right)$ if $D \leqslant_{\mathcal{Q}} D^{\prime}$ and $D^{\prime} \leqslant_{\mathcal{Q}} D$.

In practice, equivalence between databases guarantees that the answer to any query on the schema of one database can be obtained via a query on the schema of the other and vice versa, for a fixed query language. In that regard, note as well that equivalence between databases requires derivability between the corresponding database schemas in both directions.

At last, we remark that this definition is "semantic" in the sense that it is based on the idea that both databases should embody exactly the same data, rather than imposing some sort of logical/structural equality between their schemas. As such, in general, it also has the advantage of allowing for the comparison of databases that were designed and/or evolved independently.

In what follows, we show that the relation we have defined is indeed an equivalence relation.

Proposition 3.2. The notion of $\mathcal{Q}$-equivalence between databases is an equivalence relation, i.e., it satisfies the properties of reflexivity, symmetry and transitivity.

Proof. Let $\mathcal{Q}$ be a query language. To prove reflexivity, then we must show that $D \equiv_{\mathcal{Q}} D$ for every $D=(R, F) \in \mathcal{D}_{\mathcal{A}}$. By the definition of $\mathcal{Q}$-equivalence between databases, this implies showing $\mathcal{Q}$-derivability in both directions, i.e., $D \leqslant_{\mathcal{Q}} D$ and $D \leqslant_{\mathcal{Q}} D$. Yet, considering that both expressions are equal, it suffices to prove only one of them. To that end, let $R=\left(r_{1}, \ldots, r_{n}\right)$. Then, by the definition of $\mathcal{Q}$-derivability between databases, we must show that
(a) $\mathcal{S}((R, F)) \leqslant_{\mathcal{Q}} \mathcal{S}((R, F))$ by some tuple of queries $\bar{q}=\left(q_{1}, \ldots, q_{n}\right)$;
(b) $r_{i} \equiv q_{i}((R, F))$ for each $1 \leq i \leq n$.

Observe that, since the relation sets are equal in both databases, we just need to show that if $\bar{q}$ is the tuple of identity queries for each relation in $R$, i.e., $q_{i}((R, F))=r_{i}$, then (a) and (b) are true.

For (b) the result is straightforward: for very relation $r_{i}$ in $R$, we need $r_{i} \equiv q_{i}\left(\left(R, F^{\prime}\right)\right)$ to be true. Given that $q_{i}((R, F))=r_{i}$, then we have $r_{i} \equiv r_{i}$, which is clearly true.

Before showing that (a) is also true, recall that every instance of a database schema has the same relation schemas (and consequently the same number of relations). Now, let $S=\mathcal{S}((R, F))$ and $S^{\prime}=\mathcal{S}((R, F))$. Then, to prove (a), we must show that for every instance $D_{1}=\left(\left(r_{1}^{\prime}, \ldots, r_{n}^{\prime}\right), F\right)$ of $S$, there is an instance $D_{2}$ of $S^{\prime}$ such that for every relation $r_{i}^{\prime}$ we have $r_{i}^{\prime} \sqsubseteq q_{i}\left(D_{2}\right)$. However, taking into account that $q_{i}$ is always the identity query for a relation in $R_{D_{2}}$, then, instead, for each relation in $r_{i}^{\prime}$ we must have $r_{i}^{\prime} \equiv q_{i}\left(D_{2}\right)$. Furthermore, since the relation schemas of $S$ and $S^{\prime}$ are equal, then this is only true if every relation set in an instance of $S$ is also in an instance of $S^{\prime}$. In order to demonstrate this, first, recall that the relation sets in the instances of both database schemas must satisfy the closure of the respective set of FDs. Thus, since the closure of the set of FDs in $S$ and $S^{\prime}$ is equal, then the allowed instances for both database schemas are exactly the same, and therefore the last "if" condition is trivially satisfied (i.e, any relation set in an instance of $S$ is also in an instance of $S^{\prime}$ ). This, at last, proves that (a) is indeed true and that $\mathcal{Q}$-derivability between databases holds the property of reflexivity.

In order to prove symmetry, we must show that for any pair of databases $D$ and $D^{\prime}$ over $\mathcal{A}$, then $D \equiv_{\mathcal{Q}} D^{\prime}$ implies $D^{\prime} \equiv_{\mathcal{Q}} D$. By the definition of $\mathcal{Q}$-derivability between databases, we have that $D \equiv_{\mathcal{Q}} D^{\prime}$ implies $D \leqslant_{\mathcal{Q}} D^{\prime}$ and $D^{\prime} \leqslant_{\mathcal{Q}} D$, and that $D^{\prime} \equiv_{\mathcal{Q}} D$ implies $D^{\prime} \leqslant_{\mathcal{Q}} D$ and $D \leqslant_{\mathcal{Q}} D^{\prime}$. Thus, we have to prove that, together, $D \leqslant_{\mathcal{Q}} D^{\prime}$ and $D^{\prime} \leqslant_{\mathcal{Q}} D$ imply $D^{\prime} \leqslant_{\mathcal{Q}} D$ and $D \leqslant_{\mathcal{Q}} D^{\prime}$. This is clearly true, as the expressions for $\mathcal{Q}$-derivability are exactly the same. Therefore, $\mathcal{Q}$-equivalence between databases satisfies the property of symmetry.

Finally, for transitivity, we must show that for any additional database $D^{\prime \prime} \in \mathcal{D}_{\mathcal{A}}$, then $D \equiv_{Q} D^{\prime}$ and $D^{\prime} \equiv_{Q} D^{\prime \prime}$ imply $D \equiv_{Q} D^{\prime \prime}$. First, let us transform the expressions of $\mathcal{Q}$ equivalence into expressions of $\mathcal{Q}$-derivability. Therefore, we must show that $D \leqslant_{\mathcal{Q}} D^{\prime}$, $D^{\prime} \leqslant_{\mathcal{Q}} D, D^{\prime} \leqslant_{\mathcal{Q}} D^{\prime \prime}$ and $D^{\prime \prime} \leqslant_{\mathcal{Q}} D^{\prime}$ imply $D \leqslant_{\mathcal{Q}} D^{\prime \prime}$ and $D^{\prime \prime} \leqslant_{\mathcal{Q}} D$. Let us start by focusing on $D \leqslant_{\mathcal{Q}} D^{\prime \prime}$. If $D \leqslant_{\mathcal{Q}} D^{\prime}$ is true, then it must exist a tuple of queries $\bar{q}$ such that $\mathcal{S}(D) \leqslant_{\mathcal{Q}} \mathcal{S}\left(D^{\prime}\right)$ by $\bar{q}$ and, for each relation $r \in R_{D}$, we have $r \equiv q\left(D^{\prime}\right)$ for some $q \in \bar{q}$. This means that $\bar{q}$ allows us to obtain all relations of $D$ in $D^{\prime}$, and that for every instance of $\mathcal{S}(D)$ we can extract all its information in an instance of $\mathcal{S}\left(D^{\prime}\right)$ with $\bar{q}$. Furthermore, if $D^{\prime} \leqslant_{\mathcal{Q}} D^{\prime \prime}$, then it also exists a tuple of queries $\bar{q}^{\prime}$ such that $\mathcal{S}\left(D^{\prime}\right) \leqslant_{\mathcal{Q}} \mathcal{S}\left(D^{\prime \prime}\right)$ by $\bar{q}^{\prime}$ and, for each relation $r^{\prime} \in R_{D^{\prime}}$, we have $r^{\prime} \equiv q^{\prime}\left(D^{\prime \prime}\right)$, where $q^{\prime} \in \bar{q}^{\prime}$. Again, this implies that all relations of $D^{\prime}$ can be obtained in $D^{\prime \prime}$, and that for every instance of $\mathcal{S}\left(D^{\prime}\right)$ there is an instance of $\mathcal{S}\left(D^{\prime \prime}\right)$ from which we can extract at least the same information using $\bar{q}^{\prime}$. Now,
to derive $D \leqslant_{Q} D^{\prime \prime}$, then it must exist a third tuple of queries $\bar{q}^{\prime \prime}$ such that $\mathcal{S}(D) \leqslant_{\mathcal{Q}} \mathcal{S}\left(D^{\prime \prime}\right)$ by $\bar{q}^{\prime \prime}$ and, for each relation $r \in R_{D}$, we have $r \equiv q^{\prime \prime}\left(D^{\prime \prime}\right)$ for some $q^{\prime \prime} \in \bar{q}^{\prime \prime}$. Thus, since if $D \leqslant_{Q} D^{\prime}$ is true we can get all relations of $D$ in $D^{\prime}$ using $\bar{q}$, and if $D^{\prime} \leqslant_{Q} D^{\prime \prime}$ is true we can do it for all relations of $D^{\prime}$ using $D^{\prime \prime}$, then it is also true that if $\bar{q}^{\prime \prime}$ is the composition of $\bar{q}^{\prime}$ with $\bar{q}$, we can obtain all relations of $D$ in $D^{\prime \prime}$. In addition, the tuple of queries $\bar{q}^{\prime \prime}$ also guarantees that for every instance of $\mathcal{S}(D)$ there is an instance of $\mathcal{S}\left(D^{\prime \prime}\right)$ with at least the same information, as it transforms any instance of $\mathcal{S}(D)$ into one of $\mathcal{S}\left(D^{\prime}\right)$ and subsequently $\mathcal{S}\left(D^{\prime \prime}\right)$. Thus, $D \leqslant_{\mathcal{Q}} D^{\prime}$ and $D^{\prime} \leqslant_{\mathcal{Q}} D^{\prime \prime}$ imply $D \leqslant_{\mathcal{Q}} D^{\prime \prime}$.

A similar reasoning can be applied to derive $D^{\prime \prime} \leqslant_{\mathcal{Q}} D$ from $D^{\prime \prime} \leqslant_{\mathcal{Q}} D^{\prime}$ and $D^{\prime} \leqslant_{\mathcal{Q}} D$. Therefore, we proved that $\mathcal{Q}$-equivalence has the property of transitivity, completing the proof for the proposition.

To conclude this chapter, we demonstrate two important results that are a consequence of our definition of $\mathcal{Q}$-derivability between databases and will be particularly useful in the remainder of this dissertation.

The first follows from the observation that if two databases have the same relation sets and the closure of their FDs is comparable (one is either a subset or superset of the other), then we can always claim a direction for $\mathcal{Q}$-derivability. In fact, the database that is more constrained is $\mathcal{Q}$-derivable from the other, since it has less possible instances for its schema (note that $\mathcal{Q}$-derivability between databases depends on $\mathcal{Q}$-derivability between the respective schemas, whose instances are constrained by the FDs; additionally, the more imposing the FDs are, the fewer possible instances there are for the schema).

Proposition 3.3. Let $(R, F)$ and $\left(R, F^{\prime}\right)$ be two databases over $\mathcal{A}$ such that $F^{\prime+} \subseteq F^{+}$. Then, $(R, F) \leqslant Q\left(R, F^{\prime}\right)$ is true.

Proof. The proof for this proposition is very similar to the one for reflexivity in Proposition 3.2. Considering that the relation sets in both databases are equal, then we can still assume $\bar{q}$ to be the tuple of identity queries. Hence, the proof for the second condition follows directly from there. Regarding the proof for $\mathcal{Q}$-derivability between the database schemas, recall that since the relation schemas are equal, then we just need to show that every relation set in an instance of $S=\mathcal{S}((R, F))$ is also in an instance of $S^{\prime}=\mathcal{S}\left(\left(R, F^{\prime}\right)\right)$. For that, note that all relation sets in an instance of a database schema must satisfy the closure of the set of FDs of the schema. Furthermore, if the relation sets satisfy the closure of a set of FDs, then they also satisfy any subset of it. In this case, we have that the closure of the FDs in $S^{\prime}$ is at most as constrained as the closure of the FDs in $S$, i.e., $F^{\prime+} \subseteq F^{+}$. This implies that any relation set in an instance of $S$ can also be in an instance of $S^{\prime}$, given that the relations in $S$ are either more constrained than those in $S^{\prime}$ or equally constrained (never less constrained). Therefore $S \leqslant_{\mathcal{Q}} S^{\prime}$ holds and $(R, F) \leqslant_{\mathcal{Q}}\left(R, F^{\prime}\right)$ is indeed true.

The next result is based on the observation that adding a set of relations to a database does not break any relationships of $\mathcal{Q}$-derivability with the databases that derive from it. In other words, one can say that all the databases that are $\mathcal{Q}$-derivable from a particular
database $D$, remain to be $\mathcal{Q}$-derivable from it even if a new set of relations is added to $D$. This is due to the fact that the new relations do not influence the queries that guarantee $\mathcal{Q}$-derivability.

Proposition 3.4 (Monotonicity of Derivability with respect to $Q$ between Databases). Let $(R, F)$ and $\left(R^{\prime}, F^{\prime}\right)$ be two databases over $\mathcal{A}$. Then, $(R, F) \leqslant Q\left(R^{\prime}, F^{\prime}\right)$ implies $(R, F) \leqslant Q$ ( $R^{\prime} \cup R^{\prime \prime}, F^{\prime}$ ), for any $R^{\prime \prime} \subseteq \mathcal{R}_{\mathcal{A}}$.

Proof. Let $R=\left(r_{1}, \ldots, r_{n}\right)$. For $(R, F) \leqslant_{Q}\left(R^{\prime}, F^{\prime}\right)$ to be true then it must exist a tuple of queries $\bar{q}=\left(q_{1}, \ldots, q_{n}\right)$ on the schema of database $\left(R^{\prime}, F^{\prime}\right)$ that satisfies the conditions for $\mathcal{Q}$-derivability. Since any query in $\bar{q}$ corresponds to a collection of operations on a subset of the relations in ( $R^{\prime}, F^{\prime}$ ), then it is easy to see that adding an arbitrary set of relations $R^{\prime \prime} \in \mathcal{R}_{\mathcal{A}}$ to $R^{\prime}$ will not change the queries in $\bar{q}$ nor their answers. In truth, we can still say that $\bar{q}$ is defined for the schemas of the same relations in the new $\operatorname{dbs} \mathcal{S}\left(\left(R^{\prime} \cup R^{\prime \prime}, F\right)\right)$. As a consequence, $(R, F) \leqslant_{Q}\left(R^{\prime}, F^{\prime}\right)$ implies $(R, F) \leqslant Q\left(R^{\prime} \cup R^{\prime \prime}, F^{\prime}\right)$.

In the remaining of this work we shall fix $\mathcal{Q}$ to be conjunctive queries in the relational algebra, and thus drop the symbol $\mathcal{Q}$ from the notation of $\mathcal{Q}$-derivability (now derivability) between database schemas as well as $\mathcal{Q}$-derivability and $\mathcal{Q}$-equivalence (now respectively derivability and equivalence) between databases.

The advantage of this language is that it allows us to be expressive enough so that there exist equivalent databases, but not expressive to the point that (almost) all of them are equivalent to each other. Furthermore, the conjunctive queries use the operators that enable us to write the majority of the queries in real-world applications.

## Forgetting Relations in Relational Databases

Having presented the problem, overviewed related literature and introduced the concepts that serve as the basis for the work developed in this dissertation, we are now ready to discuss our proposed solution. To that end, we start by exploring a pragmatic use case, motivating the need for a theory of forgetting relations in relational databases that takes into account non-overlapping requirements for forgetting. Afterwards, we present the general definition of an operator of relation forgetting and discuss two different profiles of forgetting, corresponding to non-transitive and transitive forgetting. For each category, we present a concrete operator. Finally, we conclude the chapter by introducing two additional operators for transitive forgetting that are built upon the first to satisfy a requirement that is often indispensable: no information that was forgotten can be recovered. In between, we illustrate the operators using several intuitive examples.

### 4.1 Motivating Example

We now revisit the problem stated in the Introduction and consider the points discussed in Section 2.5 regarding the need to define forgetting operators for relational databases that comply with the 'right to be forgotten' and address situations in which data anonymisation is not possible (e.g., due to legal reasons), or undesirable (e.g., because it negatively affects processing of other legitimate information).

For that, consider Figure 4.1. The relations depicted in this figure correspond to a very simplified version of part of an insurance company database. The first relation, customer, which is exactly the same as the relation represented in Figure 3.1 from the previous chapter, stores information about the customers of the company. As we saw, this relation has the attributes ID, which uniquely identifies each customer, Name and YearOf Birth. Furthermore, the relation serious_disease indicates the serious diseases of the customers. As such, it has the attributes ID and Disease. At last, the relation increased_cost, which has the attributes Disease and Amount, gives information on the amount of money that should be added to the price of a particular health insurance plan, for each disease ${ }^{1}$.

[^14]| customer |  |  | serious_disease |  | increased_cost |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| ID | Name | YearOfBirth | ID | Disease | Disease | Amount |
| 1 | Anna | 1990 | 1 | Breast Cancer | HIV/AIDS | 600 |
| 2 | David | 1971 | 2 | Type 2 Diabetes | Breast Cancer | 500 |
| 3 | John | 1990 | 2 | Prostate Cancer | Type 2 Diabetes | 500 |
| 4 | Mary | 1987 | 3 | Lung Cancer | Lung Cancer | 650 |
|  |  |  |  |  | Prostate Cancer | 550 |

Figure 4.1: Part of the relations of an insurance company database.

Regarding the relation customer, in addition to the fact that each customer is identified by the value on the column $I D$, we assume that they have a single name and year of birth. As for the relation increased_cost, we assume that every value for the attribute Disease can only be associated with at most a single value for the attribute Amount. Therefore, the database has the functional dependencies

$$
\begin{aligned}
F= & \{I D \rightarrow \text { Name, YearOf Birth } \\
& \text { Disease } \rightarrow \text { Amount }\} .
\end{aligned}
$$

Now, assume that a court has ordered that insurance companies can no longer collect, store and process information about their customers' diseases, provided that they have been overcome or mitigated ${ }^{2}$.

Supposing that the contents of the relation serious_disease were collected in previous years, in order to comply with the court orders, the company of this example decides to withdraw all information in it. Therefore, as discussed in Section 2.5, a first approach could be to simply erase relation serious_disease. This way, the company cannot associate anymore the customers to their diseases. In that case, the resulting database is given by Figure 4.2.

| customer |  |  |
| :---: | :---: | :---: |
| ID | Name | YearOfBirth |
| 1 | Anna | 1990 |
| 2 | David | 1971 |
| 3 | John | 1990 |
| 4 | Mary | 1987 |


| increased_cost |  |
| :---: | :---: |
| Disease | Amount |
| HIV/AIDS | 600 |
| Breast Cancer | 500 |
| Type 2 Diabetes | 500 |
| Lung Cancer | 650 |
| Prostate Cancer | 550 |

Figure 4.2: Relations of the insurance company database resulting from removing the relation serious_disease from the original set.

[^15]However, with this approach, the company would also potentially lose valuable information about the prices of the plans practiced in the past for certain customers, considering that the indirect relationship between the attributes ID and Amount is lost. In a scenario where it would be impractical or even infeasible to gather that data any other way (e.g., by revisiting old contracts or asking customers directly), the company would certainly want to preserve this relationship.

In order to solve this issue, before erasing the relation serious_disease, the database administrator could apply the natural join operator ${ }^{3}$ to the relations serious_disease and increased_cost, and then project the result on the attributes ID and Amount, removing the attribute Disease. This process results in a new relation that can be added to the database under any unused name (anyway, for future reference that the relation was in fact the result of some operation between serious_disease and increased_cost, we will assume it was named disease-increased_cost). Finally, the administrator could safely remove the relation serious_disease from the database. The set of relations resulting from this proceeding are shown in Figure 4.3.

| customer |  |  | disease-increased_cost |  | increased_cost |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| ID | Name | YearOfBirth | ID | Amount | Disease | Amount |
| 1 | Anna | 1990 | 1 | 500 | HIV/AIDS | 600 |
| 2 | David | 1971 | 2 | 500 | Breast Cancer | 500 |
| 3 | John | 1990 | 2 | 550 | Type 2 Diabetes | 500 |
| 4 | Mary | 1987 | 3 | 650 | Lung Cancer | 650 |
|  |  |  |  |  | Prostate Cancer | 550 |

Figure 4.3: Relations of the insurance company database resulting from removing the relation serious_disease from the original set and adding the relation disease-increased_cost to it.

Regarding the FDs, since $I D \rightarrow$ Amount is not in $F$ and cannot be inferred by the ones that are (i.e., it does not belong to the closure of $F$ ), we can conclude that the relations in the new set satisfy $F$.

In reality, both the first approach of simply removing relation serious_disease and the procedure we have just explained can be considered very simple operators of forgetting. Evidently, the latter alternative has the advantage of preserving the indirect (transitive) relationship between the attributes ID (of relation serious_disease) and Amount (of relation increased_cost) while forgetting about relation serious_disease. From that perspective, we can say that we forgot serious_disease while keeping transitive information with respect to increased_cost.

Nevertheless, as desirable as the result of this operator might sound, it is still possible to indirectly associate some customers with their diseases, even if there is no relation that does so directly. For example, we know from the relation increased_cost, that

[^16]the only disease that has an increased cost of 650 is Lung Cancer. Thus, since the customer with ID 3 participates with amount 650 in the new relation disease-increased_cost, we know for sure that the tuple ( 3 , Lung Cancer) was in the instance of the forgotten relation serious_disease (of course, assuming that its schema is still known after forgetting the relation). Analogously, we can infer that ( 2 , Prostate Cancer) was also in the instance of serious_disease, considering that Prostate Cancer is the only value that participates with amount 550 in increased_cost and that the tuple $(2,550)$ belongs to disease-increased_cost.

In this case, to obey the court orders without having to remove more relations than the strictly needed, the company would need a "smarter" forgetting operator that could understand which tuples are possible to infer about the relation that was forgotten and act accordingly. For instance, in our example, the operator could automatically eliminate the tuples $(2,550)$ and $(3,650)$ from the new relation disease-increased_cost, since without these tuples it will no longer be possible to recover any of the customer's diseases. Note that, in these circumstances, this would be the best case scenario for the company of the example, which has to find a balance between keeping as much information as possible about older insurance plans and doing so without recovering what was forgotten.

As we will see later in the chapter, in other settings, a more viable alternative would be to add specific tuples to the new relation, instead of removing the problematic ones. This would be especially relevant in the cases where removing at least one tuple would invalidate the usefulness of the whole relation. Obviously, the tuples to be added have to be chosen in such a way that it would still not be possible to retrieve any of the original ones.

So far we have shown that there are different alternatives to forget a relation in a database. In fact, we argue that there is no single one size-fits-all operator of forgetting, but rather a large set of operators, each with its own unique characteristics, and therefore more suitable than the others for certain applications. With that in mind, in the next section, we will discuss in more detail what exactly a forgetting operator is. Subsequently, we will propose concrete operators of forgetting for the different approaches considered until now, and will show with some new examples that these operators are indeed adjustable and generalisable to any input database (and relation to be forgotten), independently of its set of relations and FDs.

### 4.2 Relation Forgetting

To address the problem of forgetting relations in the relational model, we start by formally introducing the notion of a relation forgetting operator. Then, we explore different perspectives on forgetting, some of which are motivated by the previous example. In between, we define operators that are aligned with each of these perspectives. In any case, before continuing, we draw attention to the fact that in this dissertation we will focus solely on forgetting about a single relation.

Intuitively, the main idea of forgetting about a relation in a database is that the resulting database does not include the relation to be forgotten. Moreover, when it comes to its set of FDs, we also want the resulting database to be at most as constrained as the original one, given that a relation was deleted. Following these principles, a relation forgetting operator is defined as a function that given an initial database $D$, a relation to be forgotten $\delta$ and a relation $\psi$, returns a unique database resulting from forgetting about $\delta$ in $D$ (i.e., where $\delta$ is absent from it) while possibly keeping transitive information with some of the attributes in $\psi$.

Definition 4.1 (Relation Forgetting Operator). Given a database $(R, F)$ and two relations $\delta$ and $\psi$, a relation forgetting operator is a function $\mathrm{f}: \mathcal{D}_{\mathcal{A}} \times \mathcal{R}_{\mathcal{A}} \times\left(\mathcal{R}_{\mathcal{A}} \cup\{\emptyset\}\right) \rightarrow \mathcal{D}_{\mathcal{A}}$ such that $\mathrm{f}((R, F), \delta, \psi)=\left(R^{\prime}, F^{\prime}\right)$ is a database where $\delta \notin R^{\prime}$ and $F^{\prime+} \subseteq F^{+}$. We call $\mathrm{f}((R, F), \delta, \psi)$ the result of forgetting about $\delta$ in $(R, F)$ while preserving transitive information with respect to $\psi$.

In the definition above we remain as general as possible, imposing only the minimum conditions a relation forgetting operator must satisfy. This way, our definition can accommodate different views on forgetting, in similar fashion to what is done in the related literature and in specific on forgetting in ASP. Indeed, this definition still allows for a myriad of possible operators, some of which more adequate than others, depending on the requirements of the application at hand.

Also, it is worth noting that the relation $\psi$ in the definition might not be given as input in an operator (that is, it can be the empty set). This is particularly relevant for the cases where we want to forget without transitivity and thus are only interested in the first two arguments of the operator. That being said, we will first focus on this type of forgetting. In fact, from here on, we shall call it non-transitive forgetting.

### 4.3 Non-Transitive Forgetting

As we saw in the example in Section 4.1, non-transitive forgetting is particularly useful when we want (or are obliged) to simply remove the relation to be forgotten from the database.

Moreover, in those cases, we would expect the operators that fit in this category of forgetting to preserve all information in the remaining relations. Likewise, we want them not to add any extra data to the database. Thus, ideally, the original and resulting databases are equivalent up to the relation to be forgotten $\delta$. Only this way we guarantee that, for any query on the schema of the original database without this relation, there is a corresponding query on the schema of the resulting database, and vice versa.

As for the way FDs should be handled by these operators, intuitively, in these conditions, we would expect them to keep under equivalence at least all the FDs in the original set that are projected on the schema of some relation in the database besides relation $\delta$. In addition, we also wish to preserve those that are not projected on the schema of any
relation, as they are unrelated to $\delta$. Basically, we want the closure of the initial and final sets of FDs to be equivalent up to the ones exclusively projected on the schema of $\delta$. In some cases, leaving the FDs as they are is enough to guarantee this.

Finally, an operator that fulfils these requirements is given by Algorithm 2.

```
Algorithm 2: Non-transitive relation forgetting operator \(\mathrm{f}_{n t}\)
input :Triple \((D, \delta, \psi)\) such that \(D=(R, F) \in \mathcal{D}_{\mathcal{A}}, \delta \in \mathcal{R}_{\mathcal{A}}\) and \(\psi \in \mathcal{R}_{\mathcal{A}} \cup\{\emptyset\}\)
output: Database resulting from \(\mathrm{f}_{n t}(D, \delta, \psi)\)
    \(R^{\prime} \leftarrow R \backslash\{\delta\} ;\)
    return ( \(\left.R^{\prime}, F\right)\);
```

The operator $f_{n t}$ generates the result of forgetting by simply removing the relation to be forgotten from $R$. Therefore, the resulting database is the closest possible to the original one.

In that respect, it is important to acknowledge that the definition of non-transitive forgetting does not imply that all operators should manipulate $R$ in the exact same way. In fact, it is absolutely acceptable for non-transitive operators to carry out transformations at the level of the schema of the database. As long as all information in $R \backslash\{\delta\}$ is preserved and none is added, that would still fit in our definition of non-transitive forgetting.

In addition, note that the result of the operator $\mathrm{f}_{n t}$ is independent of the relation $\psi$. This should always be the case for non-transitive forgetting in general.

Going back to the illustrative example used in Section 4.1, the result of applying the operator $\mathrm{f}_{n t}$ to forget $\delta=$ serious_disease in the database shown in Figure 4.1 is precisely the database in Figure 4.2 with the original set of FDs. Again, $\psi$ could be any relation or even the empty set.

However, as already discussed, non-transitive forgetting has an obvious disadvantage in that it does not preserve indirect relationships that are exclusively guaranteed by the relation to be forgotten. For this reason, in alternative to non-transitive forgetting, in the next section we introduce transitive forgetting. Unlike its counterpart, transitive forgetting seeks to preserve the indirect relationships that are originally assured by the relation to be forgotten.

### 4.4 Transitive Forgetting

We now turn our attention to transitive forgetting. As we saw in Section 4.1, the key idea of this type of forgetting is to preserve some of the information in the natural join between the relation to be forgotten $\delta$ and another relation in the database $\psi$, i.e., $\delta \bowtie \psi$, that would otherwise be lost using a non-transitive operator ${ }^{4}$.

[^17]Yet, before doing so, we have to make sure some conditions are met. First, we are only interested in transitivity between relations such that neither of the schemas is included in the other ${ }^{5}$, as this allows us to ignore the cases where there certainly are no indirect relationships to keep among different attributes. Naturally, this is valid both for $s(\delta)$ being a superset of $s(\psi)$ (i.e., containing all of its attributes) and vice versa. When two relations satisfy this condition we call them non-comparable.

Definition 4.2 (Non-comparability between Relations). Let $r$ and $r^{\prime}$ be two relations over $\mathcal{A}$. We say that $r$ and $r^{\prime}$ are non-comparable, denoted by $r \nsim r^{\prime}$, if $s(r) \nsubseteq s\left(r^{\prime}\right)$ and $s\left(r^{\prime}\right) \nsubseteq s(r)$.

Moreover, another requirement for transitive forgetting is that the schemas of the relations $\delta$ and $\psi$ are non-disjoint, i.e., $s(\delta) \cap s(\psi) \neq \emptyset$. If that was not the case, then there would not be any relationships between the attributes in these relations, and therefore no transitive information to preserve. In that regard, when two non-comparable relations $r$ and $r^{\prime}$ have non-disjoint schemas, we denote it by $r \nmid r^{\prime}$.

Looking back at the relations $\delta=$ serious_disease and $\psi=$ increased_cost, which motivated transitive forgetting in the database represented in Figure 4.1, it is apparent that these relations are non-comparable. Taking into account that $s($ serious_disease $)=$ (ID,Disease) and $s($ increased_cost $)=($ Disease, Amount $)$, there is exactly one attribute in each relation that does not belong to the other. Moreover, it is also obvious that the schemas of these relations are non-disjoint, since $s($ serious_disease $) \cap s($ increased_cost $)=$ Disease. For these reasons, we have serious_disease $\nmid$ increased_cost.

All things considered, if at least one of these conditions was not true, i.e., $\delta \nmid \psi$ did not hold (for instance, because $\delta$ was equal to $\psi$ or had the same schema), then we would expect any transitive operator to simply remove $\delta$ from the set of relations that compose the database, without undertaking any kind of transitivity, akin to non-transitive operators. Of course, similarly to the latter, it would still be reasonable for transitive operators to transform the remaining relations, provided that no information is lost. Additionally, if $\psi$ is not in the database (or, to the same effect, corresponds to the empty set), then we would also expect transitive operators not to perform any transitive operation.

Now that the conditions for transitivity are laid out, we can finally discuss the relation resulting from the natural join between $\delta$ and $\psi$. Henceforth, we shall call it the transitive relation.

Regarding this relation, we want the common attributes (i.e., the ones that are nondisjoint) for $\delta$ and $\psi$ to be removed from it, else one would not truly forget about $\delta$. For instance, returning to the database in Figure 4.3, the transitive relation diseaseincreased_cost, does not have the attribute Disease, which was in $\delta=$ serious_disease, because it also belonged to $\psi=$ increased_cost.

[^18]Furthermore, since we would expect operators to keep the information of $\psi$ in the resulting database, and given that that $\psi$ may have any arity, oftentimes we may want to omit some (but not all) of its attributes from the transitive relation. As such, we must accept as the result of the transitivity a relation that does not mention all the disjoint attributes of $\psi$ in its schema. In fact, this is also the case for the attributes in $\delta$ given that, ultimately, the goal is to forget about this relation. To discuss this subject in more detail, consider the following example.
Example 4.1. Consider once again the relations in Figure 4.1. Now, assume that, for instance, the relation increased_cost, which corresponds to the relation to keep transitive information with serious_disease further on in the example, had a third attribute Discount, corresponding to a percentage that will be deduced from the final price of the insurance plan. In that case, $s$ (increased_cost $)=($ Disease,_Amount,Discount $)$, and thus the transitive relation disease-increased_cost could have one of the schemas (ID,Amount), (ID, Discount) or (ID, Amount, Discount).

Regarding the example above, the choice for the schema of the transitive relation would obviously fall on the database owner/administrator, as it depends on the situation itself. A similar reasoning would also apply in case the relation serious_disease had higher arity as well.

In short, if we designate by $\theta$ the attributes that will be projected out of $\delta \bowtie \psi$, denoted $(\delta \bowtie \psi)_{\| \theta}$, then we can have $\theta \supseteq s(\delta) \cap s(\psi)$. Even so, in order to guarantee that there is, in fact, transitivity between $\delta$ and $\psi$, we have to make sure that $\theta \nsupseteq s(\delta)$ and that $\theta \nsupseteq s(\psi)$. In other words, $\theta$ must be a non-total intersecting superset of $(s(\delta), s(\psi)$ ), which is formally defined as follows.

Definition 4.3 (Non-total Intersecting Superset). Let $s$ be a set and ( $s_{1} \ldots, s_{n}$ ), with $1<n$, a tuple of sets. We say that $s$ is a non-total intersecting superset of $\left(s_{1} \ldots, s_{n}\right)$, denoted by $s ?_{\cap}\left(s_{1}, \ldots, s_{n}\right)$, if

- $s \supseteq s_{1} \cap \ldots \cap s_{n}$; and
- $s \nsupseteq s_{i}$ for each $1 \leq i \leq n$.

We highlight the importance of ensuring that the transitivity is only done when the relations $\delta$ and $\psi$ are non-comparable: if either $s(\delta)$ or $s(\psi)$ were a superset of the other, then it would not exist a non-total intersecting superset for these two relation schemas.

Proposition 4.1. Given two relations $r$ and $r^{\prime}$ over $\mathcal{A}$, if they are not non-comparable, i.e., if $r \nsim r^{\prime}$ is not true, then there is no set of attributes $s \subseteq \mathcal{A}$ such that $s \supseteq_{\cap}\left(s(r), s\left(r^{\prime}\right)\right)$.

Proof. Assume that $r \nsim r^{\prime}$ does not hold. Then, at least one of the conditions in the definition of non-comparability does not hold, i.e., we have $s(r) \subseteq s\left(r^{\prime}\right), s\left(r^{\prime}\right) \subseteq s(r)$ or both. Without loss of generality, suppose that $s(r) \subseteq s\left(r^{\prime}\right)$. Now, if $s$ is any set of attributes that
satisfies the first condition of Definition 4.3, i.e., $s \supseteq s(r) \cap s\left(r^{\prime}\right)$, then, because $s(r) \subseteq s\left(r^{\prime}\right)$, it is also true that $s \supseteq s(r)$. This, in turn, contradicts the second condition of the definition. Therefore, $s$ cannot be a non-total intersecting superset of $\left(s(r), s\left(r^{\prime}\right)\right)$.

Having discussed how transitive operators should handle the transitive relation, we now turn our attention to the FDs that should be in any database resulting from transitive forgetting. Perhaps, in this case, the first intuition could be to adopt what is done for non-transitive forgetting. This way, the operators would preserve under equivalence at least all the FDs in the original set that are not exclusively projected on the schema of the relation to be forgotten (for instance because they are projected on the schema of some other relation) as well as those that can be implied by transitivity through the ones that are only projected on its schema. Besides, this also means that the operators would keep in the resulting database the FDs that are completely independent of the attributes in the relation to be forgotten.

Nevertheless, pursuing this idea could lead to the non-satisfaction of some of the preserved FDs by the transitive relation. The next example illustrates this situation.
Example 4.2. Consider a database $D B \in \mathcal{D}_{\mathcal{A}}$ with the relations $\delta, \psi$ and $r$ shown below and the set of functional dependencies $\{A \rightarrow B ; A \rightarrow C ; B \rightarrow D ; D \rightarrow E\}$.

| $\delta$ |  |  |
| :---: | :---: | :---: |
| $A$ | $B$ | $C$ |
| $\mathrm{a}_{1}$ | $\mathrm{~b}_{1}$ | $\mathrm{c}_{1}$ |
| $\mathrm{a}_{2}$ | $\mathrm{~b}_{1}$ | $\mathrm{c}_{2}$ |
| $\mathrm{a}_{3}$ | $\mathrm{~b}_{1}$ | $\mathrm{c}_{3}$ |



If the result of forgetting about $\delta$ in $D B$ with respect to $\psi$ is a database with the relations $\psi, r$ and $\phi$, where $\phi$ corresponds to the transitive relation and has schema $(A, B, D)$ then, by following how FDs are handled by non-transitive operators, we would expect at least the functional dependencies $\{B \rightarrow D ; A \rightarrow D ; D \rightarrow E\}$ to be preserved under equivalence in the resulting database. The reasoning behind this is that $B \rightarrow D$ is not projected on the schema of any relation in the original database; $A \rightarrow D$ is obtained by transitivity using $A \rightarrow B$ (only projected on $s(\delta)$ in the original database) and $B \rightarrow D$; and $D \rightarrow E$ is projected on $s(r)$ (and is independent of the attributes in $\delta$ ). In reality, one could also argue that the functional dependency $A \rightarrow B$ must be in the resulting database, taking into account that although it is projected exclusively on the schema of $\delta$ in $D B$, it is now projected on the schema of the new relation $\phi$. Regarding the functional dependency $A \rightarrow C$, since it is projected on $s(\delta)$ but not on the schema of any other relation in the original or resulting databases, non-transitive operators would not necessarily require its preservation.

Either way, if we look more closely at the relation $\phi$, which is shown below, it is clear that it does not satisfy the functional dependencies $B \rightarrow D$ and $A \rightarrow D$, as the first and
second tuples have the same values for the attributes $A$ and $B$, but different values for the attribute $D$.

| $\phi$ |  |  |
| :---: | :---: | :---: |
| $A$ | $B$ | $D$ |
| $\mathrm{a}_{1}$ | $\mathrm{~b}_{1}$ | $\mathrm{~d}_{1}$ |
| $\mathrm{a}_{1}$ | $\mathrm{~b}_{1}$ | $\mathrm{~d}_{2}$ |
| $\mathrm{a}_{2}$ | $\mathrm{~b}_{1}$ | $\mathrm{~d}_{1}$ |

An important conclusion we can draw from the previous example is that even FDs in the original set that are not projected on the schema of a single relation may not be satisfied by the transitive one. Thus, since we are only interested in operators that are well defined (i.e., that output databases), we must strengthen the conditions of non-transitive forgetting for the FDs we want to preserve in the result of transitive forgetting. Ultimately, we need to find a balance between keeping as many of the original FDs as possible in the resulting database and ensuring that the relations satisfy those FDs. Perhaps the most obvious choice would be to not only do not consider for equivalence the FDs that are exclusively projected on the relation to be forgotten, but also those that depend on the attributes of said relation, regardless of the relation schemas where they are projected on.

Returning to Example 4.2, this would mean that both $B \rightarrow D$ and $A \rightarrow D$ would not be taken into account for equivalence between the original and resulting sets of FDs, since attributes $A$ and $B$ belong to the schema of $\delta$. Therefore, these FDs could be safely removed from the result of forgetting. Still on the same example, we would also ignore the functional dependency $A \rightarrow B$ even if it was projected on the schema of some other relation, which does not happen in the case of non-transitive forgetting. This, at last, is extremely relevant, since without one (or, in this case, both) of the functional dependencies $A \rightarrow B$ and $B \rightarrow D$, we guarantee that $A \rightarrow D$ does not take part in the equivalence condition in any way, given that it would be impossible to infer it by means of transitivity.

To further justify the need to impose this more conservative condition upon the result of transitive forgetting, note that the FDs that may lead to a set not being satisfied by the transitive relation, may not only be projected on the schema of $\delta$, but on the schema of some other relation(s) in the database (in addition to, as we saw earlier, not being projected at all). The following example shows exactly this situation.

Example 4.3. Suppose the database $D B$ in Example 4.2 had two additional relations $r_{1}$ and $r_{2}$ over $\mathcal{A}$, such that $s\left(r_{1}\right)=(A, F)$ and $s\left(r_{2}\right)=(F, D)$, and the functional dependencies $A \rightarrow F$ and $F \rightarrow D$. In this case, the FDs are projected on the schemas of the relations $r_{1}$ and $r_{2}$, respectively, and by transitivity we can infer the functional dependency $A \rightarrow D$.

Regarding the example above, neither of the FDs is projected on the schema of $\delta$, yet we must remove one of them to avoid deriving $A \rightarrow D$. In this case, it would be $A \rightarrow F$, since attribute $A$ belongs to $s(\delta)$.

In summary, the goal for transitive forgetting is that the operators keep under equivalence at least all the FDs of the original set that do not mention the attributes that belong to the relation to be forgotten $\delta$. Intuitively, this means that we are weakening the equivalence required for non-transitive forgetting in such a way that only the FDs which do not have attributes in $\delta$ are considered for equivalence. Of course, this does not imply that all the other FDs must be deleted. In reality, they can also be kept the resulting database, as long as their closure is satisfied by the relation set.

To talk about the FDs that do not refer to the attributes in $\delta$, in the next definition we introduce the concept of $\delta$-exclusion of a set of FDs.

Definition 4.4 ( $\delta$-Exclusion of a Set of Functional Dependencies). Let $F$ be a set of FDs over $\mathcal{A}$ and $\delta$ a relation over $\mathcal{A}$. The $\delta$-exclusion of $F$, denoted by $F_{\| \delta}$, corresponds to the set of all FDs in $F^{+}$that do not include attributes of $\delta$, i.e., $F_{\| \delta}=\left\{X \rightarrow Y \in F^{+} \mid X \cap s(\delta)=\right.$ $\emptyset \wedge Y \cap s(\delta)=\emptyset\}$.

In general, the notion of $\delta$-exclusion of a set of FDs $F$ cannot be computed by removing from $F^{+}$the FDs that are projected on $\delta$, i.e. $F_{\delta}$, since, by definition, this set only contains FDs such that all its attributes belong to $s(\delta)$. Naturally, this is not the case for the definition of $F_{\| \delta}$, as it excludes from $F^{+}$any FD that has at least one attribute of $\delta$, be it on the left or right-hand side. With that in mind, we introduce a procedure to compute $F_{\| \delta}$, which is given by Algorithm 3.

```
Algorithm 3: Computation of the \(\delta\)-exclusion of a set of FDs
input : Pair \((F, \delta)\) such that \(F \subseteq \mathcal{F}_{\mathcal{A}}\) and \(\delta \in \mathcal{R}_{\mathcal{A}}\)
output:Set of FDs resulting from compute- \(\delta\)-exclusion \((F, \delta)\)
    compute the closure of \(F\), denoted \(F^{+}\), using procedure in [SKS11] and initialise
    \(F^{\prime} \leftarrow F^{+} ;\)
    foreach \(X \rightarrow Y \in F^{\prime}\) do
        if \(X \cap s(\delta) \neq \emptyset\) or \(Y \cap s(\delta) \neq \emptyset\) then
            \(F^{\prime} \leftarrow F^{\prime} \backslash\{X \rightarrow Y\} ;\)
    return \(F^{\prime}\);
```

Having discussed the desirable set of relations and FDs in the result of transitive forgetting, we can finally introduce our first operator for this type of forgetting. It is given by Algorithm 4 and is named $f_{t}$.

The principal idea of the operator $\mathrm{f}_{t}$ is that it preserves transitive relationships between some of the attributes in the relation to be forgotten $\delta$ and those of another relation $\psi$ in the database, provided that the conditions for transitivity we have been discussing are satisfied. This means that, in those cases, we guarantee that all the queries on the schema of the initial database that are based on the projection of a set of attributes without those in $\theta$ over the natural join between $\delta$ and $\psi$ have always a corresponding query on the database resulting from forgetting about $\delta$.

```
    Algorithm 4: Transitive relation forgetting operator \(\mathrm{f}_{t}\)
input :Triple \((D, \delta, \psi)\) such that \(D=(R, F) \in \mathcal{D}_{\mathcal{A}}, \delta \in \mathcal{R}_{\mathcal{A}}\) and \(\psi \in \mathcal{R}_{\mathcal{A}} \cup\{\emptyset\}\)
output: Database resulting from \(\mathrm{f}_{t}(D, \delta, \psi)\)
    \(R^{\prime} \leftarrow R \backslash\{\delta\} ;\)
    \(F^{\prime} \leftarrow F ;\)
    /* conditions for transitivity */
    if \(\delta, \psi \in R\) and \(\delta \nmid \psi\) then
        \(\theta \leftarrow s(\delta) \cap s(\psi) ;\)
        \(R^{\prime} \leftarrow R^{\prime} \cup\left\{(\delta \bowtie \psi)_{\| \theta}\right\} ;\)
        \(F^{\prime} \leftarrow\) compute- \(\delta\)-exclusion \((F, \delta)\);
    return \(\left(R^{\prime}, F^{\prime}\right)\);
```

More precisely, the operator starts by removing the relation to be forgotten $\delta$ from the original set of relations $R$, assigning the resulting relation set to $R^{\prime}$. Then, it assigns the initial set of FDs to $F^{\prime}$. Subsequently, in line 3, it verifies the conditions for transitivity, i.e., whether $\delta$ and $\psi$ belong to the relation set of the initial database, are non-comparable and their schemas are non-disjoint. If that is the case, then the operator computes $\theta$, which corresponds to the attributes that will be projected out of the new transitive relation, which in turn is computed in line 5 by joining the relations $\delta$ and $\psi$. For this concrete operator, $\theta$ is simply the intersection of the schemas of the previous relations. This means that the schema of the transitive relation will have all the remaining attributes in the union of the schemas of $\delta$ and $\psi$, i.e, its schema will be $s(\delta) \cup s(\psi) \backslash(s(\delta) \cap s(\psi))$. For instance, if we look at the setting in Example 4.1, then $\theta$ would be the singleton with attribute Disease, and the schema of the transitive relation would be (ID, Amount, Discount). Continuing with the explanation of the operator, after adding the new relation to $R^{\prime}$, it assigns to the variable $F^{\prime}$ the result of the $\delta$-exclusion of $F$, which is obtained using Algorithm 3. Finally, the operator concludes by returning the database ( $R^{\prime}, F^{\prime}$ ).

Unsurprisingly, the application of the operator $\mathrm{f}_{t}$ to forget relation $\delta=$ serious_disease while keeping transitive information with respect to $\psi=$ increased_cost in the motivating example in Figure 4.1, leads precisely to the relations in Figure 4.3.

Furthermore, it is clear that, similarly to what was discussed at the time, for some instances of $\delta$ and $\psi$, the operator $\mathrm{f}_{t}$ has the inconvenience of allowing the recovery of at least some of the tuples in the instance of the relation that was (apparently) forgotten. This happens because the operator always preserves all transitive information between $\delta$ and $\psi$ (except, of course, that of the attributes in the intersection of the respective schemas). Thus, with the knowledge that the operator works in this way, one can sometimes deliberately "guess" some of the tuples in $\delta$.

Therefore, even though the operator can be useful when recovery of forgotten tuples is not problematic (e.g., in those cases where we want to simplify the database by removing some auxiliary relation while preserving transitive information), the reality is that, in specific situations, it might be highly undesirable. Hence, to overcome this drawback of
the operator $\mathrm{f}_{t}$, in the next section we present two transitive forgetting operators that are built upon it but, contrarily to $f_{t}$, do not allow intentional recovery of the tuples in $\delta$, even when it is known which operator was used to obtain the result of forgetting.

### 4.5 Refining Transitive Forgetting

As suggested before, we are mainly interested in transitive operators that do not allow deliberate recovery of forgotten tuples through the transitive relation added to the database. However, to define such operators, it is necessary to understand what it means to recover forgotten tuples and when it is, or it is not, possible to do it.

Intuitively, one way of guaranteeing that there is no recovery of forgotten tuples is to ensure that it exists an alternative to each tuple in the relation to be forgotten $\delta$ that, when exchanged with the original, leads to exactly the same database resulting from this operation. Following this principle, it will not be possible to infer any of the tuples that are in the instance of the relation, even if one knows which operator was used.

That is exactly the concept behind the operator $\mathrm{f}_{t^{-}}$, which is given by Algorithm $5^{6}$. Concretely, the operator verifies whether for each tuple $t$ in the instance of $\delta$, there is an alternative tuple $t^{*}$ such that, if we exchange these tuples in $i(\delta)$, the result of the operator remains the same. For $f_{t^{-}}$in particular, this means that the transitive relation is equal for both alternatives. Furthermore, in case $t$ has no such alternative tuple $t^{*}$, then it is simply not considered for transitivity (as a matter of fact, we can view the "empty tuple" as its alternative). Regarding this construction, it is crucial that the modified relation $\delta$ with $t^{*}$ instead of $t$ still satisfies the FDs in the database, namely those projected on $s(\delta)$. With this strategy, even if we know the operator that computed the result of forgetting, we cannot retrieve any of the tuples that were in the relation that was forgotten, as there is at least an alternative to each tuple.

To get a better understanding of the intuition behind the algorithm before we analyse it more thoroughly, let us consider the following example.

Example 4.4. Recall the insurance company database, whose relations are shown in Figure 4.1. As we saw, in this setting we want to forget about relation serious_disease while keeping transitive information with respect to the relation increased_cost. Hence, in our notation, we have $\delta=$ serious_disease and $\psi=$ increased_cost. Thus, the transitive relation resulting from joining $\delta$ and $\psi$, and projecting the result on the attributes that are not in both schemas, corresponds to the relation disease-increased_cost illustrated in Figure 4.3 (recall that this is precisely the transitive relation we would get from applying the operator $\mathrm{f}_{t}$ ). To make it easier to follow the example, consider the next tables, which represent the three relations (from left to right, we have relation $\delta$, relation $\psi$ and the transitive relation).

[^19]| serious_disease |  | increased_cost |  | disease-increased_cost |  |
| :---: | :---: | :---: | :---: | :---: | :---: |
| ID | Disease | Disease | Amount | ID | Amount |
| 1 | Breast Cancer | HIV/AIDS | 600 | 1 | 500 |
| 2 | Type 2 Diabetes | Breast Cancer | 500 | 2 | 500 |
| 2 | Prostate Cancer | Type 2 Diabetes | 500 | 2 | 550 |
| 3 | Lung Cancer | Lung Cancer | 650 | 3 | 650 |
|  |  | Prostate Cancer | 550 |  |  |

In this example, the first tuple in serious_disease, i.e., (1, Breast Cancer) will be joined with the tuple (Breast Cancer, 500) in increased_cost, resulting in the tuple $(1,500)$ in disease-increased_cost. Therefore, to make sure that having $(1,500)$ in the transitive relation will not allow us to infer (1, Breast Cancer), we need to find an alternative to this last tuple that, when joined with the tuples in increased_cost, would as well lead to $(1,500)$ in the transitive relation. The first thing we can notice by looking at this case in particular is that any alternative tuple of (1, Breast Cancer) that would lead to the same result in the transitive relation, must have value 1 for attribute $I D$. Secondly, regarding the value for attribute Disease, it is evident that it cannot be equal to Breast Cancer (as the tuple and its alternative must be different). However, it has to participate in increased_cost with exactly the same value of Amount as Breast Cancer (i.e., 500), otherwise, the tuple resulting from the join operation will not be ( 1,500 ). Thus, in this setting, the only possible alternative tuple to (1, Breast Cancer) would be (1, Type 2 Diabetes). In fact, note that this tuple can only be a valid alternative because, e.g., the functional dependency Disease $\rightarrow I D$ does not exist in the database, given that ( 2 , Type 2 Diabetes) is already in the instance of serious_disease. Regarding this last tuple, it is also easy to see that its alternative would be ( 2 , Breast Cancer). As for the next tuple in the instance of serious_disease, i.e. (2, Prostate Cancer), since there is no value of attribute Disease that participates with value 550 for Amount in increased_cost, then it does not have any alternative, and therefore would not be considered by the operator $\mathrm{f}_{t^{-}}$ for transitivity. A similar reasoning applies for the last tuple. All in all, the goal of the operator is to find all tuples in serious_disease that have valid alternatives.

Taking into account the last example, we can draw some general conclusions about an alternative tuple $t^{*}$ for each tuple $t$ in $i(\delta)$. The first such conclusion is that they must have the same value for all the attributes of $\delta$ that appear in the transitive relation, so that we can arrive at the same result in this relation. As usual, let us denote by $\theta$ the attributes that will be projected out of the transitive relation (e.g., for the example above we have $\theta=\{$ Disease $\}$ ). Then, this means that we want $t$ and $t^{*}$ to have equal values for the attributes $s(\delta) \backslash \theta$. Note that, in turn, this implies that the remaining attributes must be different for $t^{*}$ and $t$, since we want $t^{*} \neq t$. In fact, we can also come to a verdict regarding the values for these attributes, which brings us to our second and final conclusion. Observe that, to ensure that we indeed get the same result in the transitive
relation, it is essential that the values for these attributes in $t$ and $t^{*}$ participate in $\psi$ with exactly the same values for the attributes of $\psi$ that are in the transitive relation (i.e., those of $s(\psi) \backslash \theta)$. Only this way we guarantee that, regardless of whether $i(\delta)$ has $t$ or $t^{*}$, the transitive relation is precisely the same. In reality, the only thing that changes is the "path" taken to reach it.

We are now ready to analyse the operator in more detail. First of all, we draw attention to the fact that it extends $f_{t}$, and therefore some of the principles presented therein for any transitive operator in general still apply. For instance, the operator also starts by removing $\delta$ from the initial set of relations, assigning the result to $R^{\prime}$. Subsequently, it assigns the original set of FDs to variable $F^{\prime}$. In line 3, the operator verifies the conditions for transitivity. Similarly to $f_{t}$, we want $\delta$ and $\psi$ to be non-comparable relations in the initial database whose schemas are non-disjoint. If these requirements are met, then the operator computes the attributes that will be projected out of the transitive relation and assigns them to variable $\theta$. From this point on, $\mathrm{f}_{t^{-}}$has a different behaviour than the operator $f_{t}$, given that their ultimate goal is completely different. Whereas $f_{t}$ computes the transitive relation by directly joining $\delta$ and $\psi$, the new operator $\mathrm{f}_{t^{-}}$only considers for transitivity the tuples in $\delta$ that would not lead to their retrieval through the transitive relation (i.e., because they have a valid alternative tuple).

To that end, the operator starts by creating a new relation $\delta^{*}$, which is initialised with the same name and schema as $\delta$ but without any tuples. The purpose of this relation is to store the tuples in $i(\delta)$ that have an alternative and, for that reason, can be considered for transitivity. Thus, from line 6 to line 13, the operator iterates over all tuples $t$ in the instance of $\delta$, verifying whether they are suitable or not to be added to $\delta^{*}$ (which is done line 13). The first such verification (line 7) checks if the values for $\theta$ in $t$ are also in $\psi$. If they are not, then this tuple is not relevant for transitivity anyway, and thus we may proceed to the next tuple, discarding the current one. In case the values are in $\psi$, then the operator searches if it exists a tuple $t^{*}$ that can be the alternative for $t$ (line 9). The domain for the values of $t^{*}$ is straightforward: the values for $s(\delta) \backslash \theta$ must be in $\delta$ and those for $\theta$ in $\psi$. Next, through line 9 to 12 , the operator imposes four conditions on $t^{*}$. The first, still in line 9 , is that this tuple must be different from $t$. Moreover, the second and the third, in lines 10 a 11, correspond to the conclusions one and two, respectively, drawn from the preceding paragraph. Finally, the last condition is related to the FDs. Taking into consideration that we want $\delta$ to still satisfy the FDs of the original database in case $t$ is replaced with $t^{*}$, then we need to be extra careful with the values of $t^{*}$. In practice, to guarantee that the modified $\delta$ does not infringe any functional dependency $X \rightarrow Y$, then a first, perhaps more naive approach could be to force $t$ and $t^{*}$ to have the exact same values for the attributes $X$ and $Y$ (whenever they are in the schema of $\delta$ ). It is obvious that this way the new $\delta$ would still satisfy all relevant FDs. However, there are two situations in particular where this conditions is unnecessarily strong. The first is when the attributes $X$ and $Y$ are a subset of $\theta$. Considering that the domain for the values of $t^{*}$ in the attributes $\theta$ comes from the tuples in $\psi$, and that this relation already
satisfies the FDs projected on its schema, then we do not need to worry about these FDs. The second situation where we do not have to impose the equality between $t$ and $t^{*}$ is when it does not exist another tuple $t^{\prime}$ in the instance of $\delta$ whose values for the attributes in $X \backslash \theta$ coincide with those of $t$. Since $t$ and $t^{*}$ will have the same values for the attributes in $s(\delta) \backslash \theta$ (due to the equality in line 10), then they will also have the same values for $X \backslash \theta$, and given that no tuple in the relation has those same values for these attributes, then any functional dependency $X \rightarrow Y$ will not fail in $\delta$ if we exchange $t$ for $t^{*}$. We will come back to this discussion in the next chapter.

Finally, having iterated over all tuples in $i(\delta)$ and added the ones that have a valid alternative to the instance of $\delta^{*}$, the operator joins $\delta^{*}$ with $\psi$ and projects from the result the attributes in $\theta$ (line 14). Afterwards, it adds the resulting transitive relation to $R^{\prime}$. Before terminating by returning the database $\left(R^{\prime}, F^{\prime}\right)$, the operator computes the $\delta$-exclusion of $F$ (Algorithm 3), assigning the resulting set of FDs to $F^{\prime}$.

```
Algorithm 5: Transitive relation forgetting operator \(\mathrm{f}_{\mathrm{t}^{-}}\)
input :Triple \((D, \delta, \psi)\) such that \(D=(R, F) \in \mathcal{D}_{\mathcal{A}}, \delta \in \mathcal{R}_{\mathcal{A}}\) and \(\psi \in \mathcal{R}_{\mathcal{A}} \cup\{\emptyset\}\)
output: Database resulting from \(\mathrm{f}_{t^{-}}(D, \delta, \psi)\)
    \(R^{\prime} \leftarrow R \backslash\{\delta\} ;\)
    \(F^{\prime} \leftarrow F\);
    /* conditions for transitivity */
    if \(\delta, \psi \in R\) and \(\delta \nmid \psi\) then
        \(\theta \leftarrow s(\delta) \cap s(\psi) ;\)
        \(\delta^{*} \leftarrow(n(\delta), s(\delta), \emptyset) ;\)
        foreach \(t \in \delta\) do
            if \(t[\theta] \notin \Pi_{\theta}(\psi)\) then
                continue;
            if \(\exists t^{*} \in \Pi_{s(\delta) \backslash \theta}(\delta) \times \Pi_{\theta}(\psi)\) s.t. \(t \neq t^{*}\) and
            \(t[s(\delta) \backslash \theta]=t^{*}[s(\delta) \backslash \theta]\) and
            \(\Pi_{s(\psi) \backslash \theta}\left(\sigma_{\theta=t[\theta]}(\psi)\right)=\Pi_{s(\psi) \backslash \theta}\left(\sigma_{\theta=t^{*}[\theta]}(\psi)\right)\) and
            \(\forall X \rightarrow Y \in F^{+}\)s.t. \(X \cup Y \nsubseteq \theta\) and \(\exists t^{\prime} \in \delta \backslash\{t\}\) s.t. \(t^{\prime}[X \backslash \theta]=t[X \backslash \theta]\) then
                \(t[X \cup Y]=t^{*}[X \cup Y]\) then
                \(\left\lfloor\delta^{*} \leftarrow\left(n\left(\delta^{*}\right), s\left(\delta^{*}\right), i\left(\delta^{*}\right) \cup t\right) ;\right.\)
        \(R^{\prime} \leftarrow R^{\prime} \cup\left\{\left(\delta^{*} \bowtie \psi\right)_{\| \theta}\right\} ;\)
        \(F^{\prime} \leftarrow\) compute- \(\delta\)-exclusion \((F, \delta)\);
    return \(\left(R^{\prime}, F^{\prime}\right)\);
```

All things considered, although $f_{t^{-}}$may not preserve all transitive information between $\delta$ and $\psi$, it still guarantees that the information that is kept in the transitive relation cannot lead to the inference of the tuples in $\delta$. For that, it may only use a subset of the tuples in $\delta$ for transitivity. What happens in this case is that the instance of the transitive relation outputted by $\mathrm{f}_{t^{-}}$is always contained in the one in the result of the operator $f_{t}$, hence its name. This means that, sometimes, we can still partially answer to some of the queries that use the natural join between $\delta$ and $\psi$ projected on the schema of
the transitive relation, with the certainty that no forgotten tuples can be recovered.
In that respect, we highlight the fact that the operator gives special attention to the FDs that are projected on the schema of $\delta$, even though it ends up removing all of them from the database (as we saw in the last section, this is done to guarantee that the relation set satisfies the FDs in the database resulting from the transitive operation). The reason for this is that usually databases model real-world scenarios and, although the FDs may be explicitly deleted from the original database, oftentimes they can still be deduced.

In order to conclude the exposition of $f_{t^{-}}$, let us consider a slightly more complex example to show the application of the operator.

Example 4.5. Consider a database $D B \in \mathcal{D}_{\mathcal{A}}$ with the relations $\delta$ and $\psi$ shown below and the set of functional dependencies $F=\{B \rightarrow A ; D \rightarrow E\}$.

| $\delta$ |  |  | $\psi$ |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: |
| A | B | C | B | C | D |
| $\mathrm{a}_{1}$ | $\mathrm{b}_{1}$ | $\mathrm{c}_{1}$ | $\mathrm{b}_{1}$ | $\mathrm{c}_{1}$ | $\mathrm{d}_{1}$ |
| $\mathrm{a}_{2}$ | $\mathrm{b}_{2}$ | $\mathrm{c}_{3}$ | $\mathrm{b}_{1}$ | $\mathrm{c}_{1}$ | $\mathrm{d}_{2}$ |
|  |  |  | $\mathrm{b}_{1}$ | $\mathrm{c}_{2}$ | $\mathrm{d}_{1}$ |
|  |  |  | $\mathrm{b}_{1}$ | $\mathrm{c}_{2}$ | $\mathrm{d}_{2}$ |
|  |  |  | $\mathrm{b}_{1}$ | $c_{3}$ | $\mathrm{d}_{3}$ |
|  |  |  | $\mathrm{b}_{2}$ | $\mathrm{c}_{3}$ | $\mathrm{d}_{3}$ |

Then, by applying the operator $\mathrm{f}_{t^{-}}$to forget about $\delta$ with respect to $\psi$, the only tuple of $\delta$ that will be used for transitivity is ( $a_{1}, b_{1}, c_{1}$ ). In fact, the alternative for this tuple would be ( $\mathrm{a}_{1}, \mathrm{~b}_{1}, \mathrm{c}_{2}$ ), since they have the same values for $A=s(\delta) \backslash \theta$, and both $\left(\mathrm{b}_{1}, \mathrm{c}_{1}\right)$ and $\left(\mathrm{b}_{1}\right.$, $c_{2}$ ) participate with the same values for $D=s(\psi) \backslash \theta$ in $\psi$ (which are $\mathrm{d}_{1}$ and $\mathrm{d}_{2}$ ). Regarding the only FD in $F^{+}$that is projected on $s(\delta)$, i.e., $B \rightarrow A$, it forces the tuples to have equal values for $B$ and $A$, which is clearly the case.

Regarding the tuple $\left(\mathrm{a}_{2}, \mathrm{~b}_{2}, \mathrm{c}_{3}\right)$ in $\delta$, although $\left(\mathrm{b}_{1}, \mathrm{c}_{3}\right)$ participates with the same value as ( $\mathrm{b}_{2}, \mathrm{c}_{3}$ ) in $\psi$ (which corresponds to $\mathrm{d}_{3}$ ), because of the functional dependency $B \rightarrow A$, the tuple $\left(a_{2}, b_{1}, c_{3}\right)$ is not a valid alternative to $\left(a_{2}, b_{2}, c_{3}\right)$, given that their values for $B$ are not the same. Thus, the result of $f_{t^{-}}(D B, \delta, \psi)$ is the database $\left((\psi, \phi),\{D \rightarrow E\}^{+}\right)$, where $\phi$ is the transitive relation represented by the next table.

| $\phi$ |  |
| :---: | :---: |
| $A$ | $D$ |
| $\mathrm{a}_{1}$ | $\mathrm{~d}_{1}$ |
| $\mathrm{a}_{1}$ | $\mathrm{~d}_{2}$ |

In retrospective, the operator $\mathrm{f}_{t}$ would output the transitive relation $\phi^{\prime}$. In this case, just by looking at $\phi^{\prime}$ and $\psi$, it is easy to see that ( $\mathrm{a}_{2}, \mathrm{~b}_{1}, \mathrm{c}_{3}$ ) was a tuple of $\delta$. This implies that, in this example, $\mathrm{f}_{t}$ would allow the recovery of tuples that were supposedly forgotten.

| $\phi^{\prime}$ |  |
| :---: | :---: |
| $A$ | $D$ |
| $\mathrm{a}_{1}$ | $\mathrm{~d}_{1}$ |
| $\mathrm{a}_{1}$ | $\mathrm{~d}_{2}$ |
| $\mathrm{a}_{2}$ | $\mathrm{~d}_{3}$ |

In any case, as was hinted at the end of Section 4.1, in some circumstances, losing some of the information in the transitive relation, no matter how minimal, can be highly undesirable. For instance, consider the next example.

Example 4.6. Consider the relations food_allergy and prohibited_food depicted by the following tables.

| food_allergy |  | prohibited_food |  |
| :---: | :---: | :---: | :---: |
| EmployeeID | Allergen | Allergen | Food |
| 1 | lactose | additives | cake |
| 2 | gluten | gluten | cake |
| 3 | nuts | gluten | pasta |
| 4 | peanut | lactose | cake |
|  |  | nuts | cake |

In this example, we are interested in forgetting about the relation food_allergy, while preserving transitive information with respect to the relation prohibited_food, so that we can still have data about the foods that each employee is not supposed to eat, even if all the information about their allergies is deleted.

The transitive relation food_allergy-prohibited_food obtained by joining the relations above and projecting the result on the attributes EmployeeID and Food is illustrated below (note that this would be the transitive relation in the result of the first transitive operator $f_{t}$, or equivalently the relation obtained by asking each employee individually the foods that they cannot eat).

| food_allergy-prohibited_food |  |
| :---: | :---: |
| Allergen | Food |
| 1 | cake |
| 2 | cake |
| 2 | pasta |
| 3 | cake |

It is obvious that just by looking at this relation and at prohibited_food, it is possible to infer that the tuple ( 2 , gluten) was in relation food_allergy, since gluten is the only allergen that participates with the values cake and pasta in prohibited_food.

As we have been discussing in this section, the operator $f_{t^{-}}$would allow us to forget food_allergy while keeping transitive information wrt prohibited_food without it being possible to retrieve any of the tuples in the original relation, if such was the intended. However, for this case in particular, that would obviously mean losing some of the tuples in the transitive relation shown in the example. In particular, we would lose tuples (2, cake) and ( 2 , pasta), given that the tuple ( 2 , gluten) from food_allergy does not have an alternative that satisfies the conditions imposed by the operator (observe that the alternative for ( 1, lactose) could be either ( 1 , additives) and ( 1 , nuts); for ( 3 , nuts) could be one of ( 3 , additives) and ( 3 , lactose); and that ( 4 , peanut) would not even be considered for transitivity in the first place, since the allergen peanut does not appear in prohibited_food).

In the context of the example, one could argue that without all the original data about the foods that the employees are not supposed to eat, the transitive relation simply loses all its value. Therefore, to overcome this issue while still avoiding recovery of forgotten tuples, we present our last operator $\mathrm{f}_{t^{+}}$, which is given by Algorithm $6^{7}$.

If we consider $\delta$ to be the relation we want to forget, $\psi$ the one we want to preserve transitive information with respect to, and $\theta$ the attributes that will be projected out of the natural join between $\delta$ and $\psi$, then the operator $\mathrm{f}_{t^{+}}$guarantees that all information in $(\delta \bowtie \psi)_{\| \theta}$ is preserved without it being possible to recover the tuples that were forgotten in case there are no FDs with attributes of both $s(\delta) \backslash \theta$ and $\theta$ in the original database. Furthermore, if it does not exist at least two tuples in $\psi$ with different values for the attributes that are in $\theta$ (i.e., two tuples in the projection of $\theta$ in $\psi$ ), then the operator does not compute the transitive relation, since in those circumstances it would also lead to the recovery of forgotten tuples. Whereas the condition for the FDs allows us to greatly simplify the problem, the limitation for transitivity is a necessary requirement due to the very nature of the problem and will become clear later.

The idea behind the operator $\mathrm{f}_{t^{+}}$is very similar to the one behind $\mathrm{f}_{t^{-}}$in that we want to guarantee that every tuple $t$ in the instance of $\delta$ has an alternative such that, if we exchange it with $t$ in $i(\delta)$, we will get the exactly the same database (and consequently transitive relation) in the result of forgetting. On the other hand, the approach to achieve this result is considerably different. Contrarily to $f_{t^{-}}$, the operator $f_{t^{+}}$exclusively looks at the relation $\psi$ to verify whether it is suitable for transitivity without recovery of forgotten tuples. Intuitively, for $\psi$ to be a valid relation, for each combination of values for the attributes $\theta$ (or, in other words, for each tuple $t$ in the projection of $\theta$ in $\psi$ ), it must exist a set of tuples $T$ (where $t$ does not belong to) that participates with exactly the same values for the attributes in $s(\psi) \backslash \theta$. If that is the case, then, since every tuple in $\delta$ will be joined on the attributes of $\theta$ upon transitivity with $\psi$, we assure that for each such tuple there are at least two different "paths" to reach the same values of $s(\psi) \backslash \theta$ in the transitive relation,

[^20]and therefore its result can be equal to $(\delta \bowtie \psi)_{\| \theta}$. For instance, let us look again at the relations in Example 4.6. In this setting we have $\delta=$ food_allergy, $\psi=$ prohibited_food and $\theta$ is the singleton with attribute Allergen. Regarding the relation prohibited_food, it is clear that the allergens additives, lactose and nuts participate with the same values for the attribute Food $=s(\psi) \backslash \theta$, which means that they are viable alternatives to each other. Thus, for any employee participating with one (or multiple) of them in the relation food_allergy, we guarantee that it exists an alternative tuple that would lead to the same transitive relation. This, in turn, implies that we cannot infer the real ones.

Nevertheless, if $\psi$ is not a valid relation, which is in fact the case for prohibited_food in the example, since the allergen gluten is the only one that participates with both values cake and pasta for the attribute Food, then the operator searches for a minimal subset of tuples that can be added to $\psi$ in order to ensure that all of the original ones have an alternative. For this reason, the values for the new tuples have to be restricted to the values of $\theta$ and $s(\psi) \backslash \theta$ existing in $\psi$. Let us denote by $\psi^{\prime}$ the relation resulting from the addition of those tuples to $\psi$. Then, in practice, the operator is successively testing a subset of all the possible tuples that can be added to $\psi$ (starting from the minimal set to the maximal one) until it finds a relation $\psi^{\prime}$ that satisfies the conditions discussed, and for that reason, can be joined with $\delta$ to output the transitive relation (notice that the relation $\psi^{\prime}$ chosen is always independent of $\delta$ ). To make sure that the operator is determinist for any input (and thus it can be a function of forgetting), we impose a lexicographic order to the non-comparable sets (note that it could be any order, as long as it was always the same).

Now, regarding the algorithm to compute the operator $f_{t^{+}}$, similarly to $f_{t^{-}}$, it extends $f_{t}$, which means that it also shares some characteristics with the latter. In particular, these include the initialisation of the variables $R^{\prime}$ and $F^{\prime}$ (lines 1 and 2 ), the verification of the conditions for transitivity (line 3) and, in case those are satisfied, the computation of $\theta$ (line 4 ) and the $\delta$-exclusion of the original set of FDs $F$ (line 17). All of the remaining operations correspond to the ones discussed in detail immediately above. In line 5, the operator computes the set of all relevant tuples that can be added to $\psi$, and stores them in relation $r$. These do not include the ones in $\psi$ and, as we saw, are constrained to the values of $\theta$ and $s(\psi) \backslash \theta$ that appear in $\psi$. Furthermore, in line 6 , the power set of the instance of $r$ is computed, and in line 7 it is ordered following the aforementioned rules. From lines 8 to 18 the operator searches for a set of tuples $i$ (starting with the empty set), that can be added to the instance of $\psi$ to make it valid for transitivity without recovery of forgotten tuples (we denote this new relation by $\psi^{\prime}$ ). For that to happen, then each tuple $t$ in the projection of $\theta$ in $\psi^{\prime}$ must have an alternative set of tuples $T \backslash\{t\}$ that participates with exactly the same values for $s\left(\psi^{\prime}\right) \backslash \theta$. In specific, this condition is checked in line 12 . Finally, once a valid $\psi^{\prime}$ is found, it is joined with $\delta$ and the attributes of $\theta$ are removed from the result (observe that, in the limit, the relation $\psi^{\prime}$ with all the tuples in $r$ will be valid for transitivity, as long as $r$ has at least one tuple).

We emphasise the fact that, by the definition of the operator, the transitive relation

```
Algorithm 6: Transitive relation forgetting operator \(\mathrm{f}_{t^{+}}\)
input :Triple \((D, \delta, \psi)\) such that \(D=(R, F) \in \mathcal{D}_{\mathcal{A}}, \delta \in \mathcal{R}_{\mathcal{A}}\) and \(\psi \in \mathcal{R}_{\mathcal{A}} \cup\{\emptyset\}\)
output: Database resulting from \(\mathrm{f}_{t^{+}}(D, \delta, \psi)\)
    \(R^{\prime} \leftarrow R \backslash\{\delta\} ;\)
    \(F^{\prime} \leftarrow F ;\)
    /* conditions for transitivity */
    if \(\delta, \psi \in R\) and \(\delta \nmid \psi\) then
        \(\theta \leftarrow s(\delta) \cap s(\psi) ;\)
        \(r \leftarrow \Pi_{\theta}(\psi) \times \Pi_{s(\psi) \backslash \theta}(\psi)-\psi ;\)
        pset \(\leftarrow 2^{i(r)}\) with tuples in each set ordered by RowId
        ordered_pset \(\leftarrow\) pset ordered from minimal to maximal set, with non
        comparable sets ordered lexicographically starting with the tuples with
        lowest RowId.
        foreach \(i \in\) ordered_pset do
            \(\psi^{\prime} \leftarrow(n(\psi), s(\psi), i(\psi) \cup i) ;\)
            valid_ \(\psi^{\prime} \leftarrow\) true;
            foreach \(t \in \Pi_{\theta}\left(\psi^{\prime}\right)\) do
                if \(\nexists T \subseteq \Pi_{\theta}\left(\psi^{\prime}\right) \backslash\{t\}\) s.t. \(\Pi_{s\left(\psi^{\prime}\right) \backslash \theta}\left(\sigma_{\theta=t}\left(\psi^{\prime}\right)\right)=\Pi_{s\left(\psi^{\prime}\right) \backslash \theta}\left(\sigma_{\theta=T}\left(\psi^{\prime}\right)\right)\) then
                valid_ \(\psi^{\prime} \leftarrow\) false;
                break;
            if valid_ \(\psi^{\prime}\) then
                \(R^{\prime} \leftarrow R^{\prime} \cup\left\{\left(\delta \bowtie \psi^{\prime}\right)_{\| \theta}\right\} ;\)
                \(F^{\prime} \leftarrow\) compute- \(\delta\)-exclusion \((F, \delta)\);
                break;
    return \(\left(R^{\prime}, F^{\prime}\right) ;\)
```

in its output will always have at least as many tuples as $(\delta \bowtie \psi)_{\| \theta}$. Evidently, in certain situations, this implies that some of the information that is in the transitive relation was not originally in the database. However, it is exactly that information that allows us not to recover any of the tuples in the instance of $\delta$. Anyway, despite its obvious advantages in some settings, notice that the operator $\mathrm{f}_{t^{+}}$would not deal adequately with the example of the insurance company discussed at the beginning of the chapter (see Figure 4.1), considering that, to avoid retrieval of tuples, it would add incorrect amounts to some of the customers.

When it comes to the FDs, recall that since the relation $\psi^{\prime}$ will not be added to the result of forgetting, we do not have to worry about their satisfaction by $\psi^{\prime}$. As for the FDs in the transitive relation, we need to be extra careful with those between the attributes of $s(\delta) \backslash \theta$ and the ones of $s(\psi) \backslash \theta$, given that the FDs exclusively with attributes of one of these sets are already guaranteed to be satisfied, either because of $\delta$ (in the first case) or $\psi$ (in the second), as the values for these attributes are restricted to the ones in the respective relations. Thus, by following the more conservative approach of applying the $\delta$-exclusion to the initial set of FDs, we assure that the transitive relation does not infringe
any FD.
Let us return to Example 4.6, where $\mathrm{f}_{t^{+}}$is indeed useful. Taking into account that the relation $\psi=$ prohibited_food is not valid for transitivity, given that the allergen gluten is the only participating with both values cake and pasta in prohibited_food, the operator would create a new temporary relation prohibited_food', which would have the tuples of the original plus a minimal set of tuples that, together with the first, would guarantee transitivity without recovery of information. In this case, it could be any one of the tuples (additives, pasta), (lactose, pasta) and (gluten, pasta), but due to the fixed order imposed by the operator, it would end up choosing the first. Now, notice that all allergens in prohibited_food' have an alternative: additives and lactose participate with cake and pasta, and gluten and nuts with cake. Therefore, even though the relation resulting from the transitive operation between food_allergy and prohibited_food' would remain the same as in the example, by knowing that $\mathrm{f}_{t^{+}}$operates this way, we would not be able to infer the exact tuples that were in the instance of food_allergy after it is forgotten.

Finally, to show why $\mathrm{f}_{t^{+}}$requires at least two tuples in the projection of $\theta$ in $\psi$ to compute the transitive relation, consider the relations shown in Figure 4.4. For these instances, if the operator outputted a transitive relation with the tuple ( $\mathrm{a}_{1}, \mathrm{~d}_{1}$ ), then it would obviously lead to the recovery of all the information in $\delta$, since there are no values for the attributes $\theta=\{B, C\}$ that can be added to $\psi$ to serve as alternative for $\left(b_{1}, c_{1}\right)$.


Figure 4.4: Arbitrary relations $\delta$ and $\psi$, each with a single tuple.

Having introduced our forgetting operators, in the next chapter we will carefully analyse them and discuss their properties.

## Analysis and Evaluation

In the previous chapter we formally defined the notion of a forgetting operator and introduced four operators, each aligned with a distinct view on forgetting and thus with its own unique features and possible applications. These, in turn, were showcased using different examples and use cases.

In this chapter we analyse more thoroughly our operators of forgetting. To this end, we start by showing that all the operators are well defined, i.e., that they are indeed operators of forgetting. Afterwards, we formalise in the form of properties some of the intuitions that guided us through the definition of each of the four operators and evaluate them against those properties (with special attention to the ones that each operator was intuitively motivated by). In specific, we prove analytically which operators satisfy which properties and, when that is not the case, provide counterexamples to show the opposite. Furthermore, we prove certain relationships between some of the properties. Finally, we finish with a summary and discussion of the main results, showing that the operators obey desirable sets of the defined properties.

### 5.1 Relation Forgetting Operators

In this section we prove that our operators are correct. Therefore, for each operator individually, we show that (1) it has the correct domain and codomain; (2) the relation to be forgotten does not belong to the relation set in any possible resulting database; and (3) the closure of the FDs in any database in the output of the operator is a subset of the closure of the initial set (or, put differently, the new database is less constrained than the initial one).

Proposition 5.1. $\mathrm{f}_{n t}$ is a relation forgetting operator.
Proof. Let $\mathrm{f}_{n t}(D, \delta, \psi)=\left(R^{\prime}, F^{\prime}\right)$ for any $D=(R, F) \in \mathcal{D}_{\mathcal{A}}, \delta \in \mathcal{R}_{\mathcal{A}}$ and $\psi \in \mathcal{R}_{\mathcal{A}} \cup\{\emptyset\}$. For $\mathrm{f}_{n t}$ to be a relation forgetting operator then following statements must hold:
(1) $\mathrm{f}_{n t}$ is a function with domain $\mathcal{D}_{\mathcal{A}} \times \mathcal{R}_{\mathcal{A}} \times\left(\mathcal{R}_{\mathcal{A}} \cup\{\emptyset\}\right)$ and codomain $\mathcal{D}_{\mathcal{A}}$;
(2) $\delta \notin R^{\prime}$;
(3) $F^{\prime+} \subseteq F^{+}$.

Starting with statement (1), the domain of $\mathrm{f}_{n t}$ is $\mathcal{D}_{\mathcal{A}} \times \mathcal{R}_{\mathcal{A}} \times\left(\mathcal{R}_{\mathcal{A}} \cup\{\emptyset\}\right)$ by definition. To prove that the codomain of the operator is $\mathcal{D}_{\mathcal{A}}$, then we must show that, for any database in the output, its set of relations satisfies the closure of the FDs. This can be proved as follows. Since $(R, F) \in \mathcal{D}_{\mathcal{A}}$, then the relations in $R$ satisfy the FDs in $F^{+}$, which means that any subset of $R$ also satisfies $F^{+}$. Now, given that by definition $\mathrm{f}_{n t}(D, \delta, \psi)=(R \backslash\{\delta\}, F)$ and that $R \backslash\{\delta\}$ satisfies $F^{+}$, then $\mathrm{f}_{n t}(D, \delta, \psi)$ is a database, i.e., $\mathrm{f}_{n t}(D, \delta, \psi) \in \mathcal{D}_{\mathcal{A}}$.

Statements (2) and (3) follow directly from the fact that $\mathrm{f}_{n t}(D, \delta, \psi)=(R \backslash\{\delta\}, F)$ and, in addition for the last statement, that the Armstrong's axioms, which are used to compute the closure, are sound and complete.

Proposition 5.2. $\mathrm{f}_{t}$ is a relation forgetting operator.
Proof. Let $\mathrm{f}_{t}(D, \delta, \psi)=\left(R^{\prime}, F^{\prime}\right)$ for any $D=(R, F) \in \mathcal{D}_{\mathcal{A}}, \delta \in \mathcal{R}_{\mathcal{A}}$ and $\psi \in \mathcal{R}_{\mathcal{A}} \cup\{\emptyset\}$. For $\mathrm{f}_{t}$ to be a relation forgetting operator then following statements must hold:
(1) $\mathrm{f}_{t}$ is a function with domain $\mathcal{D}_{\mathcal{A}} \times \mathcal{R}_{\mathcal{A}} \times\left(\mathcal{R}_{\mathcal{A}} \cup\{\emptyset\}\right)$ and codomain $\mathcal{D}_{\mathcal{A}}$;
(2) $\delta \notin R^{\prime}$;
(3) $F^{\prime+} \subseteq F^{+}$.

Let us start with the first statement. The domain of $\mathrm{f}_{t}$ is $\mathcal{D}_{\mathcal{A}} \times \mathcal{R}_{\mathcal{A}} \times\left(\mathcal{R}_{\mathcal{A}} \cup\{\emptyset\}\right)$ by definition. To prove that the codomain of the operator is $\mathcal{D}_{\mathcal{A}}$, then we must show that, for any database in the output, its set of relations satisfies the closure of its FDs. To this end, note that, by the definition of $\mathrm{f}_{t}$, the resulting database is either equal to $(R \backslash\{\delta\}, F)$ or $\left(R \backslash\{\delta\} \cup\left\{(\delta \bowtie \psi)_{\| \theta}\right\}, F_{\| \delta}\right)$, where $\theta=s(\delta) \cap s(\psi)$. Regarding the first database, it was already shown to belong to $\mathcal{D}_{\mathcal{A}}$ in the proof for Proposition 5.1. So, we focus on the second. In this case, considering that $R \backslash\{\delta\}$ satisfies $F^{+}$from the previous result, then it also satisfies any subset of $F^{+}$. In particular, it satisfies $F_{\| \delta}$. Therefore, we just have to prove that the relation resulting from the join operation (i.e., the transitive relation) also satisfies all FDs in $F_{\| \delta}$ that are projected on its schema. Recall that the schema of this relation has only attributes of $\delta$ and $\psi$. Thus, since, by definition, $F_{\| \delta}$ has no FDs with attributes of $\delta$, then the only FDs that might be projected on schema of the new relation are (a subset of) the ones that are also projected on the schema of $\psi$ (bear in mind that all the FDs that could be projected on the schema of the transitive relation besides these would have attributes of $\delta$ ). Because $\psi \in R \backslash\{\delta\}$, then it satisfies all the FDs of $F_{\| \delta}$ that are projected on its schema and, by Theorem 1 in [CBN20], it follows that the transitive relation, which results from a join operation using $\psi$, also does. For these reasons, $D^{\prime} \in \mathcal{D}_{\mathcal{A}}$.

It is straightforward to see that statements (2) and (3) hold, considering that $D^{\prime}$ is either equal to $(R \backslash\{\delta\}, F)$ or $\left.\left(R \backslash\{\delta\} \cup\{\delta \bowtie \psi\}_{\| \theta}\right\}, F_{\| \delta}\right)$ and, in addition for the last statement, note that the Armstrong's axioms, which are used to compute the closure in the $\delta$-exclusion of $F$, are sound complete, and that $F_{\| \delta}$ is a subset of $F^{+}$by definition.

Proposition 5.3. $f_{t^{-}}$is a relation forgetting operator.
Proof. Let $\mathrm{f}_{t^{-}}(D, \delta, \psi)=\left(R^{\prime}, F^{\prime}\right)$ for any $D=(R, F) \in \mathcal{D}_{\mathcal{A}}, \delta \in \mathcal{R}_{\mathcal{A}}$ and $\psi \in \mathcal{R}_{\mathcal{A}} \cup\{\emptyset\}$. For $\mathrm{f}_{t^{-}}$to be a relation forgetting operator then following statements must hold:
(1) $\mathrm{f}_{t^{-}}$is a function with domain $\mathcal{D}_{\mathcal{A}} \times \mathcal{R}_{\mathcal{A}} \times\left(\mathcal{R}_{\mathcal{A}} \cup\{\emptyset\}\right)$ and codomain $\mathcal{D}_{\mathcal{A}}$;
(2) $\delta \notin R^{\prime}$;
(3) $F^{\prime+} \subseteq F^{+}$.

Starting with (1), the domain of $\mathrm{f}_{t^{-}}$is $\mathcal{D}_{\mathcal{A}} \times \mathcal{R}_{\mathcal{A}} \times\left(\mathcal{R}_{\mathcal{A}} \cup\{\emptyset\}\right)$ by definition. In order to prove that the codomain of the operator is $\mathcal{D}_{\mathcal{A}}$, then we must show that, for any database in the output, its set of relations satisfies the closure of the FDs. To that end, observe that, by the definition of the operator, the resulting database is either equal to $(R \backslash\{\delta\}, F)$ or $\left(R \backslash\{\delta\} \cup\left\{\left(\delta^{*} \bowtie \psi\right)_{\| \theta\}}, F_{\| \delta}\right)\right.$, where $\delta^{*} \sqsubseteq \delta$ and $\theta=s(\delta) \cap s(\psi)$ by construction. Considering that the first database was already shown to belong to $\mathcal{D}_{\mathcal{A}}$ in the proof for Proposition 5.1, we focus on the second. In this case, since $R \backslash\{\delta\}$ satisfies $F^{+}$from the previous result, then it also satisfies any subset of it, in particular $F_{\| \delta}$. This means that we just need to prove that the relation resulting from the join operation between $\delta^{*}$ and $\psi$ (i.e., the transitive relation) satisfies the FDs in $F_{\| \delta}$ that are projected on its schema. Recall that, by the definition of the operator (Algorithm 5, lines 5 and 13), we have $s\left(\delta^{*}\right)=s(\delta)$, and therefore the schema of the transitive relation only has attributes of $\delta$ and $\psi$. Thus, given that $F_{\| \delta}$ does not have FDs with attributes of $\delta$, the only FDs that might be projected on the schema of the new relation are either the ones that are also projected on the schema of $\psi$ or a subset of those (bear in mind that all the remaining FDs that could be projected on the schema of this relation would have attributes of $\delta$ ). Because $\psi \in R \backslash\{\delta\}$, it satisfies the FDs of $F_{\| \delta}$ that are projected on its schema. Therefore, it follows from Theorem 1 in [CBN20] that the new relation also does, since it results from a join operation using $\psi$. Thus, $D^{\prime} \in \mathcal{D}_{\mathcal{A}}$.

Concerning statements (2) and (3), it is easy to see that they hold, given that $D^{\prime}$ is either equal to $(R \backslash\{\delta\}, F)$ or $\left.\left(R \backslash\{\delta\} \cup\left\{\delta^{*} \bowtie \psi\right\}_{\| \theta}\right\}, F_{\| \delta}\right)$ and, in addition for the last statement, note that the Armstrong's axioms, which are used to compute the closure in the $\delta$-exclusion of $F$, are sound complete, and that $F_{\| \delta}$ is a subset of $F^{+}$by definition.

Proposition 5.4. $f_{t^{+}}$is a relation forgetting operator.
Proof. Let $\mathrm{f}_{t^{+}}(D, \delta, \psi)=\left(R^{\prime}, F^{\prime}\right)$ for any $D=(R, F) \in \mathcal{D}_{\mathcal{A}}, \delta \in \mathcal{R}_{\mathcal{A}}$ and $\psi \in \mathcal{R}_{\mathcal{A}} \cup\{\emptyset\}$. For $\mathrm{f}_{t^{+}}$to be a relation forgetting operator then following statements must hold:
(1) $\mathrm{f}_{t^{+}}$is a function with domain $\mathcal{D}_{\mathcal{A}} \times \mathcal{R}_{\mathcal{A}} \times\left(\mathcal{R}_{\mathcal{A}} \cup\{\emptyset\}\right)$ and codomain $\mathcal{D}_{\mathcal{A}}$;
(2) $\delta \notin R^{\prime}$;
(3) $F^{\prime+} \subseteq F^{+}$.

Regarding statement (1), the domain of $f_{t^{+}}$is $\mathcal{D}_{\mathcal{A}} \times \mathcal{R}_{\mathcal{A}} \times\left(\mathcal{R}_{\mathcal{A}} \cup\{\emptyset\}\right)$ by definition. To prove that the codomain of the operator is $\mathcal{D}_{\mathcal{A}}$, then we must show that, for any database in the output, its set of relations satisfies the closure of the FDs. Therefore, consider the two possible databases in the output of the operator, $(R \backslash\{\delta\}, F)$ and $\left(R \backslash\{\delta\} \cup\left\{\left(\delta \bowtie \psi^{\prime}\right)_{\| \theta}\right\}, F_{\| \delta}\right)$, where $\psi \sqsubseteq \psi^{\prime} \sqsubseteq\left(\Pi_{\theta}(\psi) \times \Pi_{s(\psi) \backslash \theta}(\psi)\right)$ and $\theta=s(\delta) \cap s(\psi)$ by construction. Taking into account that $(R \backslash\{\delta\}, F)$ was already shown to belong to $\mathcal{D}_{\mathcal{A}}$ in the proof for Proposition 5.1, we focus exclusively on the second database. In this case, since $R \backslash\{\delta\}$ satisfies $F^{+}$from the previous result, then it is true that it also satisfies any subset of it, in particular $F_{\| \delta}$. Thus, we just need to prove that the relation resulting from the natural join between $\delta$ and $\psi^{\prime}$ (i.e., the transitive relation) satisfies the FDs in $F_{\| \delta}$ that are projected on its schema. Recall that, by the definition of the operator (Algorithm 6, line 9), we have $s(\psi)=s\left(\psi^{\prime}\right)$, which implies that the schema of the transitive relation only has attributes of $\delta$ and $\psi$. Therefore, given that $F_{\| \delta}$ has no FDs with attributes of $\delta$, the only FDs that might be projected on the schema of the new relation are either the ones projected on the schema of $\psi$ or a subset of those (bear in mind that all the remaining FDs that could be projected on the schema of this relation would have attributes of $\delta$ ). In fact, if we look closely at the schema of the transitive relation, it only has the attributes $s(\psi) \backslash \theta$ of $\psi$. Hence, it suffices to focus on the proof for the satisfaction of the FDs that are projected on these attributes. Furthermore, note that, by the definition of the algorithm to compute $\mathrm{f}_{t^{+}}$(lines 5-9), all combinations of values in $\psi^{\prime}$ for the attributes $s(\psi) \backslash \theta$ are already in $\psi$, i.e., $\Pi_{s\left(\psi^{\prime}\right) \backslash \theta}\left(\psi^{\prime}\right) \sqsubseteq \Pi_{s(\psi) \backslash \theta}(\psi)$. This means that, since $\psi$ satisfies the FDs in $F_{\| \delta}$ that only have attributes of $s(\psi) \backslash \theta$ (given that $\psi \in R \backslash\{\delta\}$, which was shown to satisfy $\left.F_{\| \delta}\right)$, then $\psi^{\prime}$ also does. Consequently, by Theorem 1 in [CBN20], the transitive relation must also satisfy those FDs, since it is the result of a join operation using $\psi^{\prime}$. Thus, $D^{\prime} \in \mathcal{D}_{\mathcal{A}}$.

It is straightforward to see that statements (2) and (3) hold, considering that $D^{\prime}$ is either equal to $(R \backslash\{\delta\}, F)$ or $\left.\left(R \backslash\{\delta\} \cup\left\{\delta \bowtie \psi^{\prime}\right\}_{\| \Theta}\right\}, F_{\| \delta}\right)$ and, in addition for the last statement, note that the Armstrong's axioms, which are used to compute the closure in the $\delta$-exclusion of $F$, are sound complete, and that $F_{\| \delta}$ is a subset of $F^{+}$by definition.

### 5.2 Relation Forgetting Properties

In Chapter 4, we introduced the definition of a relation forgetting operator and commented on the fact that, because it was such a general definition, it purposely allowed for countless operators. For that reason, throughout that chapter, we discussed in an informal way desired characteristics the result of forgetting should have for specific situations, in order to narrow the scope of the definition to useful operators, which were later presented.

Despite that, we still lack a more formal method to evaluate the various operators. Therefore, in this section, we will revisit the intuitions that lead to each operator and, as a means to better distinguish between the different alternatives to forgetting relations, we will introduce a variety of properties. In turn, each of these properties imposes an
additional set of restrictions to the definition of an operator and, consequently, has its usefulness in a particular range of applications. Furthermore, as we saw in the last last chapter, it is the combination of different properties that motivates distinct operators of forgetting. Again, this means that there is no one-size-fits-all approach to forgetting and, ultimately, the choice of operator to use in each situation will depend on the set of properties considered most relevant in the case under discussion.

On that premisse, we start by introducing two properties that impose semantic restrictions based on the definition of derivability between databases upon the result of forgetting.

The first property is named Persistence and was informally presented in the context of both non-transitive and transitive operators. It focuses on the fact that forgetting about a relation $\delta$ in a database should not affect the information stored in the remaining relations (i.e., that information should persist after forgetting). In other words, we would expect the result of forgetting to preserve the answer to any queries (on the schema of the original database) that do not mention the relation to be forgotten. Going back to the notion of derivability between databases, this is achieved when the original database without relation $\delta$ is derivable from the new database.

Definition 5.1 (Persistence (P)). A relation forgetting operator f satisfies Persistence if, for each $D=(R, F) \in \mathcal{D}_{\mathcal{A}}, \delta \in \mathcal{R}_{\mathcal{A}}$ and $\psi \in \mathcal{R}_{\mathcal{A}} \cup\{\emptyset\}$, we have $(R \backslash\{\delta\}, F) \leqslant f(D, \delta, \psi)$.

Satisfying ( $\mathbf{P}$ ) can thus mean the addition of any set of relations to the new database, as long as, according to the definition of an operator, the relation to be forgotten is not included in it. Furthermore, it also allows for any transformation that changes the schema of the remaining relations provided that it does not lead to any loss of information in the database. For these reasons, one can think of this property as a lower bound for the information contained in the database resulting from forgetting. Naturally, we would expect any desirable operator to satisfy ( $\mathbf{P}$ ), which is indeed the case for our operators.

Proposition 5.5. $f_{n t}, f_{t}, f_{t^{-}}$and $f_{t^{+}}$satisfy ( P ).
Proof. We start with the first operator. To prove that $\mathrm{f}_{n t}$ satisfies ( $\mathbf{P}$ ), let $D=(R, F), \delta$ and $\psi$ be as in the definition of the property. Then, by the definition of $\mathrm{f}_{n t}$, we have $\mathrm{f}_{n t}(D, \delta, \psi)=(R \backslash\{\delta\}, F)$ and thus we must show that $(R \backslash\{\delta\}, F) \leqslant(R \backslash\{\delta\}, F)$. This, in turn, is a consequence of Proposition 3.2. Therefore, $\mathrm{f}_{n t}$ satisfies ( $\mathbf{P}$ ).

In the case of $f_{t}$, we must prove that for $D=(R, F), \delta$ and $\psi$ as above, it is true that $(R \backslash\{\delta\}, F) \leqslant \mathrm{f}_{t}(D, \delta, \psi)$. By definition, $\mathrm{f}_{t}(D, \delta, \psi)$ can output one of the databases $(R \backslash\{\delta\}, F)$ and $\left(R \backslash\{\delta\} \cup\left\{(\delta \bowtie \psi)_{\| \theta\}}, F_{\| \delta}\right)\right.$, where $\theta=s(\delta) \cap s(\psi)$, and thus we must show that $(R \backslash\{\delta\}, F)$ is derivable from both, i.e., $(R \backslash\{\delta\}, F) \leqslant(R \backslash\{\delta\}, F)$ and $(R \backslash\{\delta\}, F) \leqslant\left(R \backslash\{\delta\} \cup\left\{(\delta \bowtie \psi)_{\| \theta}\right\}, F_{\| \delta}\right)$. The proof for the first expression is a consequence of Proposition 3.2. Regarding the second derivability, note that by Proposition 3.3 we have $(R \backslash\{\delta\}, F) \leqslant\left(R \backslash\{\delta\}, F_{\| \delta}\right)$ since $\left(F_{\| \delta}\right)^{+} \subseteq F^{+}$by the definition of $F_{\| \delta}$. Thus, using Proposition 3.4, we can conclude that $(R \backslash\{\delta\}, F) \leqslant\left(R \backslash\{\delta\} \cup\left\{(\delta \bowtie \psi)_{\| \theta\}}\right\}, F_{\| \delta}\right)$. Therefore, $\mathrm{f}_{t}$ satisfies (P).

Let $D=(R, F), \delta$ and $\psi$ be as above. To prove that $\mathrm{f}_{t^{-}}$satisfies $(\mathbf{P})$, then we must show that for any database in the output of the operator, we have $(R \backslash\{\delta\}, F) \leqslant \mathrm{f}_{t^{-}}(D, \delta, \psi)$. By definition, $\mathrm{f}_{t^{-}}(D, \delta, \psi)$ is either equal to $(R \backslash\{\delta\}, F)$ or $\left(R \backslash\{\delta\} \cup\left\{\left(\delta^{*} \bowtie \psi\right)_{\| \theta\}}, F_{\| \delta}\right)\right.$, where $\delta^{*} \sqsubseteq \delta$ and $\theta=s(\delta) \cap s(\psi)$. Thus, we must show that $(R \backslash\{\delta\}, F) \leqslant(R \backslash\{\delta\}, F)$ and $(R \backslash\{\delta\}, F) \leqslant(R \backslash\{\delta\} \cup$ $\left.\left\{\left(\delta^{*} \bowtie \psi\right)_{\| \theta}\right\}, F_{\| \delta}\right)$. The first expression is a consequence of Proposition 3.2, and therefore true. So we focus on the second. By Proposition 3.3, we have $(R \backslash\{\delta\}, F) \leqslant\left(R \backslash\{\delta\}, F_{\| \delta}\right)$ since $\left(F_{\| \delta}\right)^{+} \subseteq F^{+}$by the definition of $F_{\| \delta}$. Thus, using Proposition 3.4, we can conclude that $(R \backslash\{\delta\}, F) \leqslant\left(R \backslash\{\delta\} \cup\left\{\left(\delta^{*} \bowtie \psi\right)_{\| \theta\}}\right\}, F_{\| \delta}\right)$, which proves that $\mathrm{f}_{t^{-}}$satisfies (P).

For the same domains of $D=(R, F), \delta$ and $\psi$ as above, we now prove that $\mathrm{f}_{t^{+}}$satisfies $(\mathbf{P})$. To that end, observe that, by the definition of the operator, $\mathrm{f}_{t^{+}}(D, \delta, \psi)$ is either equal to $(R \backslash\{\delta\}, F)$ or $\left(R \backslash\{\delta\} \cup\left\{\left(\delta \bowtie \psi^{\prime}\right)_{\| \theta}\right\}, F_{\| \delta}\right)$, where $\psi \sqsubseteq \psi^{\prime}$ and $\theta=s(\delta) \cap s(\psi)$. Therefore we must show that $(R \backslash\{\delta\}, F)$ is derivable from both databases, i.e., $(R \backslash\{\delta\}, F) \leqslant(R \backslash\{\delta\}, F)$ and $(R \backslash\{\delta\}, F) \leqslant\left(R \backslash\{\delta\} \cup\left\{\left(\delta \bowtie \psi^{\prime}\right)_{\| \theta\}}\right\}, F_{\| \delta}\right)$. Considering that the first expression is a consequence of Proposition 3.2, we focus on the second. By Proposition 3.3 we have $(R \backslash$ $\{\delta\}, F) \leqslant\left(R \backslash\{\delta\}, F_{\| \delta}\right)$ since $\left(F_{\| \delta}\right)^{+} \subseteq F^{+}$by the definition of $F_{\| \delta}$. Thus, using Proposition 3.4, we can conclude that $(R \backslash\{\delta\}, F) \leqslant\left(R \backslash\{\delta\} \cup\left\{\left(\delta \bowtie \psi^{\prime}\right)_{\| \theta}\right\}, F_{\| \delta}\right)$. Therefore, $\mathrm{f}_{t^{+}}$satisfies (P), which finishes the proof for the proposition.

Inversely, the next property, which we name Weakening, enforces an upper bound. In this case, we may expect a database resulting from an operation of relation forgetting to carry no more information than what it is possible to retrieve from the original database. This suggests that any answer to a query on the schema of the new database should not hold information that is not accessible through a query on the initial one. Hence, one can say that forgetting may weaken the database.

Definition 5.2 (Weakening (W)). A relation forgetting operator f satisfies Weakening if, for each for each $D \in \mathcal{D}_{\mathcal{A}}, \delta \in \mathcal{R}_{\mathcal{A}}$ and $\psi \in \mathcal{R}_{\mathcal{A}} \cup\{\emptyset\}$, we have $f(D, \delta, \psi) \leqslant D$.

The only operator satisfying ( W ), however, is $\mathrm{f}_{n t}$.
Proposition 5.6. $\mathrm{f}_{n t}$ satisfies $(\mathbf{W})$, but $\mathrm{f}_{t}, \mathrm{f}_{t^{-}}$and $\mathrm{f}_{t^{+}}$do not.
Proof. We start by proving that $\mathrm{f}_{n t}$ satisfies (W). To that end, let $D=(R, F) \in \mathcal{D}_{\mathcal{A}}, \delta \in \mathcal{R}_{\mathcal{A}}$ and $\psi \in \mathcal{R}_{\mathcal{A}} \cup\{\emptyset\}$. Since by the definition of $\mathrm{f}_{n t}$ we have $\mathrm{f}_{n t}(D, \delta, \psi)=(R \backslash\{\delta\}, F)$, then we must show that $(R \backslash\{\delta\}, F) \leqslant(R, F)$. This can be done using the results in Proposition 3.2 and Proposition 3.4 as follows. First, consider that, as a consequence of Proposition 3.2, we have $(R \backslash\{\delta\}, F) \leqslant(R \backslash\{\delta\}, F)$. Then, by Proposition 3.4, we can derive $(R \backslash\{\delta\}, F) \leqslant(R, F)$. Therefore, $\mathrm{f}_{n t}$ satisfies ( W ).

To prove that $\mathrm{f}_{t}$ does not satisfy $(\mathbf{W})$, consider a database $D$ with the relations customer, disease and increased_cost as shown in Figure 4.1 and the functional dependencies

$$
F=\{I D \rightarrow \text { Name, YearOf Birth } ; \text { Disease } \rightarrow \text { Amount }\} .
$$

The result of $\mathrm{f}_{t}(D$, disease, increased_cost $)$, corresponds to the database $D^{\prime}$ with the relations customer, disease and disease-increased_cost as shown in Figure 4.3 and without any FD (note that all FDs in the original set have attributes of $\delta=$ disease and therefore $F_{\| \text {disease }}=\emptyset$ ). Let us assume that $\mathrm{f}_{t}$ satisfies $(\mathbf{W})$. For this to be true, then any database in the output of the operator must be derivable from the original database. In particular, $D^{\prime} \leqslant D$ must be true. For that, there must exist a tuple of queries $\bar{q}$ in our language such that $\mathcal{S}\left(D^{\prime}\right) \leqslant \mathcal{S}(D)$ by $\bar{q}$ and, for any relation $r$ in $D^{\prime}$, we have $r \equiv q(D)$, with $q \in \bar{q}$. Let us look at the relation customer in both databases. Since the instance for this relation is equal in both, and the attributes Name and YearOf Birth are not in any other relation, then, for this relation in $D^{\prime}, q$ must be the identity query for customer in $D$, or any query equivalent to it. However, given that the instances of $S=\mathcal{S}(D)$ are more constrained than those of $S^{\prime}=\mathcal{S}\left(D^{\prime}\right)$, as the latter does not have any FD , then it is straightforward that, for example for an instance of customer in a database in $\mathcal{D}\left(S^{\prime}\right)$ with two distinct values of Name for the same value $I D$, there is no query $q$ that can represent that information in an instance of $\mathcal{D}(S)$ while guaranteeing customer $\equiv q(D)$ for the instances of the relations in $D$ and $D^{\prime}$ shown in Figures 4.1 and 4.3, respectively, since $I D \rightarrow$ Name is in $D$ and consequently in any instance of $S$. This implies that $\mathcal{S}\left(D^{\prime}\right) \leqslant \mathcal{S}(D)$ by $\bar{q}$ is not true, which contradicts the fact that $\mathrm{f}_{t}$ satisfies $(\mathbf{W})$.

To prove that the operators $f_{t^{-}}$and $f_{t^{+}}$do not satisfy $(W)$, we can use exactly the same database $D$ as the proof above, considering that, for this input database, relation $\delta$ and relation $\psi$, the output of the operators $\mathrm{f}_{t^{-}}$and $\mathrm{f}_{t^{+}}$only differs from the one of $\mathrm{f}_{t}$ in the instance of disease-increased_cost. Thus, employing the same strategy as before, we can reach a contradiction that $f_{t^{-}}$and $f_{t^{+}}$satisfy $(W)$.

The result above allows us to conclude that the intuition behind (W) enforcing a desirable upper bound on the information contained in a database resulting from forgetting is not correct for all the cases, as our operators of transitive forgetting do not satisfy this property. In fact, as shown in the proof, this is due to the FDs in the result of the operators being less imposing than the original, given that, at least for $\mathrm{f}_{t}$, it is straightforward that all the information in any database in its output can be obtained by a query on the schema of the original database. Therefore, since derivability between databases requires derivability between the respective schemas, because of how the transitive operators handle the FDs in case the conditions for transitivity are satisfied, in some circumstances it is impossible to guarantee derivability from the schema of the original database to the schema of the new database, whose instances can now accommodate more information. In truth, even though we have taken a more conservative approach when it comes to the manipulation of the FDs by the transitive operators, as examples 4.2 and 4.3 show, in some situations this manipulation is indeed necessary to guarantee that the relations in the databases in the output of the operators satisfy the FDs. This implies that $(\mathbf{W})$ is too strong for transitive forgetting in general.

Furthermore, there are other, more general, properties that guided the definition of
our operators. For instance, when forgetting about a relation $\delta$ that does not belong to the original database $D$, we would expect the database to be left unchanged under equivalence. The following property expresses this notion of forgetting being irrelevant to $\delta \notin R_{D}$.

Definition 5.3 (Irrelevance (I)). A relation forgetting operator f satisfies Irrelevance if, for each $D \in \mathcal{D}_{\mathcal{A}}$ and $\psi \in \mathcal{R}_{\mathcal{A}} \cup\{\emptyset\}$, if $\delta \notin R_{D}$, then $f(D, \delta, \psi) \equiv D$.

Once more, (I) admits transformations at the level of the database schema, as long as the original and resulting databases stay equivalent.

Proposition 5.7. $f_{n t}, f_{t}, f_{t^{-}}$and $f_{t^{+}}$satisfy ( $I$ ).
Proof. We start by showing that $\mathrm{f}_{n t}$ satisfies (I). For that, let $D=(R, F) \in \mathcal{D}_{\mathcal{A}}, \delta \in \mathcal{R}_{\mathcal{A}} \backslash R$ and $\psi \in \mathcal{R}_{\mathcal{A}} \cup\{\emptyset\}$. By the definition of $\mathrm{f}_{n t}$ we have $\mathrm{f}_{n t}(D, \delta, \psi)=(R \backslash\{\delta\}, F)$, and thus we must prove that $(R \backslash\{\delta\}, F) \equiv(R, F)$. However, since we have $R \backslash\{\delta\}=R$, this equivalence implies $(R, F) \equiv(R, F)$, which is clearly true by Proposition 3.2 (equivalence between databases holds the property of reflexivity). Therefore, we can conclude that $\mathrm{f}_{n t}$ obeys (I).

We now look at the remaining operators concurrently. Let $D=(R, F), \delta$ and $\psi$ be as above. For $\mathrm{f}_{t}, \mathrm{f}_{t^{-}}$and $\mathrm{f}_{t^{+}}$to satisfy (I), then we must prove that $\mathrm{f}_{t}(D, \delta, \psi) \equiv(R, F)$, $\mathrm{f}_{t^{-}}(D, \delta, \psi) \equiv(R, F)$ and $\mathrm{f}_{t^{+}}(D, \delta, \psi) \equiv(R, F)$, respectively, for the defined domains. Before looking at the equivalences, observe that, for $\delta \notin R$ the result of the three operators is always the database ( $R \backslash\{\delta\}, F$ ), which is obviously equal to ( $R, F$ ) since, again, $\delta$ is not in $R$. Thus, all we have to prove is that $(R, F) \equiv(R, F)$. This result is a consequence of Proposition 3.2. Hence, it is true that the transitive operators $\mathrm{f}_{t}, \mathrm{f}_{t^{-}}$and $\mathrm{f}_{t^{+}}$satisfy (I).

Anyhow, we are usually interested in forgetting about a relation that is part of the database. For that, we saw that there are two main alternatives, corresponding to nontransitive and transitive forgetting. We now focus on the first.

Recall from the introduction of non-transitive forgetting at the start of Section 4.3, the importance of guaranteeing that these operators output databases that are equivalent to the initial database up to the relation to be forgotten $\delta$. This way, we can make sure that they preserve all information in the original database without $\delta$ and, as importantly, do not add information to new database that was not already there. However, it is still the fact that we wrap this idea under our notion of database equivalence that enables operators to change the schema of the remaining relations in the database. In order to help us introduce a property of forgetting that describes exactly this, we start by formally defining the concept of equivalence up to $\delta$ between databases.

Definition 5.4 (Equivalence up to $\delta$ between Databases). Let $D=(R, F)$ and $D^{\prime}=\left(R^{\prime}, F^{\prime}\right)$ be two databases over $\mathcal{A}$ and $\delta$ a relation over $\mathcal{A}$. We say that $D$ and $D^{\prime}$ are equivalent up to $\delta$ (or $\delta$-equivalent for short), denoted by $D \equiv \delta D^{\prime}$, if $(R \backslash\{\delta\}, F) \equiv\left(R^{\prime} \backslash\{\delta\}, F^{\prime}\right)$.

As the name suggests, the notion of $\delta$-equivalence between databases is weaker than simple equivalence, given that it relaxes the conditions imposed on the latter by not considering relation $\delta$ on both sides.

By itself, $\delta$-equivalence between databases captures the essence behind non-transitive forgetting. In this case, we want the database resulting from forgetting about a relation $\delta$ to be $\delta$-equivalent to the original database. In that respect, notice that, by the definition of an operator of relation forgetting, $\delta$ cannot belong to the new database. Overall, the fact that this condition on the result of forgetting is set under a notion of semantic equivalence makes non-transitive forgetting accommodate more than just simple "delete" or "remove" operators.

Definition 5.5 (No Transitivity (NT)). A relation forgetting operator f satisfies No Transtivity if, for each $D \in \mathcal{D}_{\mathcal{A}}, \delta \in \mathcal{R}_{\mathcal{A}}$ and $\psi \in \mathcal{R}_{\mathcal{A}} \cup\{\emptyset\}$, we have $f(D, \delta, \psi) \equiv_{\delta} D$.

An operator that obeys (NT) guarantees that, for any query on the schema of the original database that does not mention the relation to be forgotten, there is a corresponding query on the schema of the resulting database that gives exactly the same answer. This is also true for the other way around.

Furthermore, we can show that satisfying (NT) implies satisfying all the properties introduced so far, i.e., (P), (W) and (I) ${ }^{1}$.

Proposition 5.8. (NT) implies (P), (W) and (I).
Proof. Let $\mathrm{f}(D, \delta, \psi)=\left(R^{\prime}, F^{\prime}\right)$ for any relation forgetting operator $\mathrm{f}, \mathrm{D}=(R, F) \in \mathcal{D}_{\mathcal{A}}, \delta \in$ $\mathcal{R}_{\mathcal{A}}$ and $\psi \in \mathcal{R}_{\mathcal{A}} \cup\{\emptyset\}$. If (NT) holds then we have $\left(R^{\prime}, F^{\prime}\right) \equiv_{\delta}(R, F)$. By definition of $\delta$-equivalence, this can be written as $\left(R^{\prime} \backslash\{\delta\}, F^{\prime}\right) \equiv(R \backslash\{\delta\}, F)$. In turn, this equivalence implies derivability in both directions, i.e., $\left(R^{\prime} \backslash\{\delta\}, F^{\prime}\right) \leqslant(R \backslash\{\delta\}, F)$ and $(R \backslash\{\delta\}, F) \leqslant$ ( $R^{\prime} \backslash\{\delta\}, F^{\prime}$ ). Regarding the latter expression, since by the definition of an operator of forgetting $\delta \notin R^{\prime}$, we obtain $(R \backslash\{\delta\}, F) \leqslant\left(R^{\prime}, F^{\prime}\right)$, which is exactly the definition of $(\mathbf{P})$.

To prove (W), we focus instead on the first expression of derivability, i.e., $\left(R^{\prime} \backslash\{\delta\}, F^{\prime}\right) \leqslant$ $(R \backslash\{\delta\}, F)$. Because $\delta \notin R^{\prime}$ by the definition of an operator of forgetting, this implies $\left(R^{\prime}, F^{\prime}\right) \leqslant(R \backslash\{\delta\}, F)$. Thus, by Proposition 3.4, it follows immediately that $\left(R^{\prime}, F^{\prime}\right) \leqslant(R, F)$, which corresponds to the definition of $(\mathbf{W})$.

Finally, in the case of ( $\mathbf{I}$ ), we go back to the equivalence $\left(R^{\prime} \backslash\{\delta\}, F^{\prime}\right) \equiv(R \backslash\{\delta\}, F)$. For the same reason as before, this can be written as $\left(R^{\prime}, F^{\prime}\right) \equiv(R \backslash\{\delta\}, F)$. Furthermore, for $\delta \notin R$, which corresponds to the domain for which $\delta$ is defined in (I), this equivalence is equal to $\left(R^{\prime}, F^{\prime}\right) \equiv(R, F)$, which leads us to the definition of the property.

Having derived (P), (W) and (I) from (NT), we proved the proposition.

[^21]Even though we have been doing so informally, from here on, we shall call any operator that satisfies (NT) a non-transitive operator. Accordingly, from the four operators introduced, only $f_{n t}$ satisfies this property.

Proposition 5.9. $\mathrm{f}_{n t}$ satisfies (NT), but $\mathrm{f}_{t^{\prime}} \mathrm{f}_{t^{-}}$and $\mathrm{f}_{t^{+}}$do not.
Proof. We start by proving that $\mathrm{f}_{n t}$ satisfies (NT). For that, let $D=(R, F) \in \mathcal{D}_{\mathcal{A}}, \delta \in \mathcal{R}_{\mathcal{A}}$ and $\psi \in \mathcal{R}_{\mathcal{A}} \cup\{\emptyset\}$. Now, we must show that $\mathrm{f}_{n t}(D, \delta, \psi) \equiv_{\delta} D$. Since $\mathrm{f}_{n t}(D, \delta, \psi)=(R \backslash\{\delta\}, F)$ by the definition of the operator, we can rewrite the $\delta$-equivalence as $(R \backslash\{\delta\}, F) \equiv_{\delta}(R, F)$, which implies $(R \backslash\{\delta\}, F) \equiv(R \backslash\{\delta\}, F)$. This, in turn, is true by Proposition 3.2. Thus, $\mathrm{f}_{n t}$ satisfies (NT).

To prove that the operators $\mathrm{f}_{t}, \mathrm{f}_{t^{-}}$and $\mathrm{f}_{t^{+}}$do not satisfy (NT) consider a database $D$ such that $R_{D}=(\delta, \psi)$, where $\delta$ and $\psi$ correspond to the relations represented by the tables right below, and $F_{D}=\{A \rightarrow B ; B \rightarrow C\}$.


Then, for the three operators, the result of forgetting about $\delta$ with respect to $\psi$ in database $D$ is the database $D^{\prime}$ such that $R_{D^{\prime}}=(\psi, \phi)$, where $\phi$ is the transitive relation depicted by the table below, and $F_{D^{\prime}}=\emptyset$.


We can easily see that $D$ and $D^{\prime}$ are not equivalent up to $\delta$, given that $R_{D^{\prime}} \backslash\{\delta\}$ has relation with the value $\mathrm{a}_{1}$ for attribute $A$, but $R_{D} \backslash\{\delta\}$ does not even have a relation with attribute $A$. This implies that $\mathrm{f}_{t}, \mathrm{f}_{t^{-}}$and $\mathrm{f}_{t^{+}}$do not satisfy (NT).

Note, however, that (NT) alone does not enforce sufficiently strong restrictions to the set of FDs we want to preserve in the result of non-transitive forgetting. In fact, in Section 4.3, we expressed our desire for the set of FDs in the database resulting from the application of this type of operators to be equivalent to the initial set up to the FDs solely projected on the schema of $\delta$. This idea leads us to the property of Strong Preservation of Funcional Dependencies, as shown below.

Definition 5.6 (Strong Preservation of Functional Dependencies (SPFD)). A relation forgetting operator f satisfies Strong Preservation of Functional Dependencies if, for each $D=(R, F) \in \mathcal{D}_{\mathcal{A}}, \delta \in \mathcal{R}_{\mathcal{A}}$ and $\psi \in \mathcal{R}_{\mathcal{A}} \cup\{\emptyset\}$ with $f(D, \delta, \psi)=\left(R^{\prime}, F^{\prime}\right)$, we have $F^{\prime+} \backslash\left(F^{\prime}\right)_{\delta}^{R^{\prime}} \equiv$ $F^{+} \backslash F_{\delta}^{R}$.

The property (SPFD) establishes a minimum set of FDs in the result of forgetting, whose closure is now constrained between the original set with and without the FDs that are only projected on $s(\delta)$, i.e., $F^{+} \backslash F_{\delta}^{R}$ and $F^{+}$, respectively. Recall that the upper limit is given by the definition of an operator of forgetting.

For this reason, (SPFD) still gives operators some freedom to manipulate the original set of FDs. Nonetheless, as extensively discussed in Section 4.4, this property can sometimes be too strong for transitive forgetting, in the sense that it would not allow us to define correct operators.

Proposition 5.10. $\mathrm{f}_{n t}$ satisfies (SPFD), but $\mathrm{f}_{t}, \mathrm{f}_{t^{-}}$and $\mathrm{f}_{t^{+}}$do not.
Proof. To prove that $\mathrm{f}_{n t}$ satisfies (SPFD), let $\mathrm{f}_{n t}(D, \delta, \psi)=\left(R^{\prime}, F^{\prime}\right)$ for any $D=(R, F), \delta$ and $\psi$ as in the definition of the property. Now, we must show that $F^{\prime+} \backslash\left(F^{\prime}\right)_{\delta}^{R^{\prime}} \equiv F^{+} \backslash F_{\delta}^{R}$. By the definition of $\mathrm{f}_{n t}$, we have $\left(R^{\prime}, F^{\prime}\right)=(R \backslash\{\delta\}, F)$, and therefore we can rewrite the last equivalence as $F^{+} \backslash F_{\delta}^{R \backslash\{\delta\}} \equiv F^{+} \backslash F_{\delta}^{R}$. Since $F_{\delta}^{R \backslash\{\delta\}}$ is equal to $F_{\delta}^{R}$ by definition, then the equivalence is true and the property holds.

For the negative results, consider the databases $D=((\delta, \psi),\{A \rightarrow B ; B \rightarrow C\})$ and $D^{\prime}=((\psi, \phi), \emptyset)$ in the proof for Proposition 5.9 (recall that $D^{\prime}$ is the database resulting from forgetting about $\delta$ with respect to $\psi$ in $D$ using any of the operators $\mathrm{f}_{t}, \mathrm{f}_{t^{-}}$and $\left.\mathrm{f}_{t^{+}}\right)$. In this case, we have $B \rightarrow C \in F_{D}^{+} \backslash\left(F_{D}\right)_{\delta}^{R_{D}}$, but since $F_{D^{\prime}}=\emptyset$, then $B \rightarrow C \notin F_{D^{\prime}}^{+} \backslash\left(F_{D^{\prime}}\right)_{\delta}^{R_{D^{\prime}}}$. Therefore, $f_{t}, f_{t^{-}}$and $f_{t^{+}}$do not satisfy (SPFD).

On that note, we now shift the focus to transitive forgetting, and introduce a property that characterises how transitive operators should manipulate the set of FDs in the database. In light of the previous property, we name it weak Preservation of Functional Dependencies.

Definition 5.7 (weak Preservation of Functional Dependencies (wPFD)). A relation forgetting operator f satisfies weak Preservation of Functional Dependencies if, for each $D=(R, F) \in \mathcal{D}_{\mathcal{A}}, \delta \in \mathcal{R}_{\mathcal{A}}$ and $\psi \in \mathcal{R}_{\mathcal{A}} \cup\{\emptyset\}$ with $f(D, \delta, \psi)=\left(R^{\prime}, F^{\prime}\right)$, we have that $F_{\| \delta}^{\prime} \equiv F_{\| \delta}$.

As expected from the discussion in Section 4.4 concerning the FDs in the result of transitive forgetting, (wPFD) weakens the equivalence enforced by (SPFD) in the sense that less FDs are potentially considered for equivalence. In practice, this property captures the intuition that operators should keep under equivalence at least all the FDs that do not refer to the attributes in the relation to be forgotten. That is, for an operator to satisfy (wPFD), it has to guarantee that the closure of the FDs in the resulting database is equivalent to the closure of the initial set, up to the ones that have attributes of $\delta$.

Still, even though the condition imposed by (wPFD) in the result of forgetting is different from the one enforced by (SPFD), some non-transitive operators that satisfy the latter can still obey (wPFD). That is exactly the case for $\mathrm{f}_{n t}$. In addition, all transitive operators also satisfy (wPFD).

Proposition 5.11. $f_{n t}, f_{t}, f_{t^{-}}$and $f_{t^{+}}$satisfy (wPFD).

Proof. The fact that $\mathrm{f}_{n t}$ satisfies (wPFD) follows directly from the observations that, by the definition of the operator, we have $F_{f_{n t}(D, \delta, \psi)}=F$ for any $D=(R, F), \delta$ and $\psi$ in the domains defined in the property, and the Armstrong's axioms, which are used to compute $F^{+}$in the $\delta$-exclusion of $F$, are sound and complete.

We now prove this result simultaneously for the transitive operators. Let $D=(R, F), \delta$ and $\psi$ be as above. To prove that $\mathrm{f}_{t}, \mathrm{f}_{t^{-}}$and $\mathrm{f}_{t^{+}}$satisfy ( $\mathbf{w P F D}$ ), we must show that $F_{\| \delta}^{\prime} \equiv F_{\| \delta}$ for any possible $F^{\prime}$ in the database resulting from forgetting about $\delta$ with respect to $\psi$ in $D$. By direct observation, we can see that $F^{\prime}$ can either be equal to $F$ or $F_{\| \delta}$ for the three operators. Considering that the result for $F$ was already shown to be true in the proof for $\mathrm{f}_{n t}$, then we focus on the second set of FDs. Thus, we must show that $F_{\| \delta} \equiv F_{\| \delta}$. Again, it is clear that these sets are indeed equivalent, given that the Armstrong's axioms, which are used to compute the closure of $F$ in the $\delta$-exclusion of $F$, are sound complete. Thereby, $f_{t}, f_{t^{-}}$and $f_{t^{+}}$satisfy (wPFD).

Regarding the set of relations in the result of transitive forgetting, recall that right at the beginning of Section 4.4, we specified the conditions for when a new transitive relation resulting from forgetting about $\delta$ with respect to a relation $\psi$ should be added to the database. More precisely, we discussed that it should only happen in case $\psi$ belongs to the initial set of relations, and $\delta$ and $\psi$ are non-comparable and their schemas nondisjoint, i.e., $\delta \nmid \psi$. Otherwise, we would expect transitive operators to just remove $\delta$ from the database.

The next property formalises this requirement. We name it $\psi$-Irrelevance, as the operators that satisfy it are irrelevant to any $\psi \in \mathcal{R}_{\mathcal{A}}$ that does not obey at least one of the aforementioned conditions.

Definition $5.8(\psi$-Irrelevance $(\psi$-I) ). A relation forgetting operator f satisfies $\psi$-Irrelevance if, for each $D \in \mathcal{D}_{\mathcal{A}}$ and $\delta \in \mathcal{R}_{\mathcal{A}}$, if $\psi \notin R_{D}$ or $\delta \nmid \psi$ is not true, then $f(D, \delta, \psi) \equiv_{\delta} D$.

Note that this property is very similar to (NT). In fact, $(\psi-\mathrm{I})$ simply restricts the domains for which (NT) is defined, so that $\delta$-equivalence is only required for the exact values where transitivity is not desirable. Therefore, any operator that satisfies (NT) obeys ( $\psi-\mathrm{I}$ ) as well.

Proposition 5.12. (NT) implies ( $\psi-\mathrm{I}$ ).
Proof. Let f be any relation forgetting operator. For f to satisfy (NT), then for each $D \in$ $\mathcal{D}_{\mathcal{A}}, \delta \in \mathcal{R}_{\mathcal{A}}$ and $\psi \in \mathcal{R}_{\mathcal{A}} \cup\{\emptyset\}$, we have $f(D, \delta, \psi) \equiv_{\delta} D$. Considering that $(\psi-\mathbf{I})$ is just a special case of (NT), in the sense that it simply restricts the domains of $\delta$ and $\psi$ for which the $\delta$-equivalence is defined, then it is straightforward that if f satisfies (NT) it also satisfies ( $\psi-\mathbf{I}$ ).

Proposition 5.13. $\mathrm{f}_{n t}, \mathrm{f}_{t}, \mathrm{f}_{t^{-}}$and $\mathrm{f}_{t^{+}} \operatorname{satisfy}(\psi-\mathrm{I})$.

Proof. The fact that $\mathrm{f}_{n t}$ satisfies $(\psi-\mathrm{I})$ is a corollary of Propositions 5.9 and 5.12.
In the case of the transitive operators, let $D=(R, F), \delta$ and $\psi$ be as in the definition of $(\psi-I)$. To prove that $f_{t}, f_{t^{-}}$and $f_{t^{+}}$satisfy this property, we must show that if $\psi \notin R$ or $\delta \nmid \psi$ is not true, then $\mathrm{f}_{t}(D, \delta, \psi) \equiv_{\delta} D, \mathrm{f}_{t^{-}}(D, \delta, \psi) \equiv_{\delta} D$ and $\mathrm{f}_{t^{+}}(D, \delta, \psi) \equiv_{\delta} D$, respectively. Observe that, since the conditions for transitivity are not satisfied, the three operators always output the database $(R \backslash\{\delta\}, F)$, and thus it suffices to prove that $(R \backslash\{\delta\}, F) \equiv_{\delta}(R, F)$. By the definition of $\delta$-equivalence, this implies $(R \backslash\{\delta\}, F) \equiv(R \backslash\{\delta\}, F)$, which is true by Proposition 3.2. Therefore, $\mathrm{f}_{t}, \mathrm{f}_{t^{-}}$and $\mathrm{f}_{t^{+}}$hold $(\psi-\mathrm{I})$.

Contrarily to $(\psi-\mathbf{I})$, whenever the requirements for transitivity between $\delta$ and $\psi$ are fulfilled, we want our transitive operators to add the new transitive relation to the database. The next two properties, $\left(\mathbf{T}^{-}\right)$and $\left(\mathbf{T}^{+}\right)$, capture the idea of transitive forgetting. The first property guarantees that there is transitivity even if only a subset of the tuples in the instance of $(\delta \bowtie \psi)_{\| \theta}$, where $\theta$ is a non-total intersecting superset of $(s(\delta), s(\psi))$, is preserved. On the other hand, the second is similar, but for a superset. The distinction between these two properties is extremely important. As we saw in Section 4.5, in some cases, it is either by avoiding some tuples in the instance of $(\delta \bowtie \psi)_{\| \theta}$, or adding tuples to it, that we can preserve the indirect relationship between some of the disjoint attributes in $\delta$ and $\psi$ without it being possible to use this information to deliberately retrieve any of the forgotten tuples.

Definition 5.9 (Transitivity with Deletion ( $\mathrm{T}^{-}$)). A relation forgetting operator f satisfies Transitivity with (possible) Deletion if, for each $D=(R, F) \in \mathcal{D}_{\mathcal{A}}$ and $\delta, \psi \in R$ with $\delta \nmid \psi$, there exists $\phi \in R_{f(D, \delta, \psi)} \backslash R$ such that $\phi \sqsubseteq(\delta \bowtie \psi)_{\| \theta}$ for some $\theta \supseteq_{\cap}(s(\delta), s(\psi))$.

Proposition 5.14. $\mathrm{f}_{t}$ and $\mathrm{f}_{t^{-}}$satisfy $\left(\mathrm{T}^{-}\right)$, but $\mathrm{f}_{n t}$ and $\mathrm{f}_{t^{+}}$do not.
Proof. We start by proving that $\mathrm{f}_{t}$ satisfies $\left(\mathbf{T}^{-}\right)$. For that, let $D=(R, F) \in \mathcal{D}_{\mathcal{A}}, \delta \in R$ and $\psi \in R$. If $\delta \nmid \psi$ holds, then by the definition of the operator, we have $\mathrm{f}_{t}(D, \delta, \psi)=$ $\left(R \backslash\{\delta\} \cup\left\{(\delta \bowtie \psi)_{\left.\| \theta^{*}\right\}}, F_{\| \delta}\right)\right.$, where $\theta^{*}=s(\delta) \cap s(\psi)$. Therefore, $R_{f_{t}(D, \delta, \psi)} \backslash R$ is equal to $\left(R \backslash\{\delta\} \cup\left\{(\delta \bowtie \psi)_{\| \theta^{*}}\right\}\right) \backslash R$, which in turn is equal to $(\delta \bowtie \psi)_{\| \theta^{*}}$. Now, consider $\theta^{*}$. Our goal is to prove that it is a non-total intersection superset of $(s(\delta), s(\psi))$, i.e., $\theta^{*} \supseteq_{\cap}(s(\delta), s(\psi))$. For that, $\theta^{*} \supseteq s(\delta) \cap s(\psi), \theta^{*} \nsupseteq s(\delta)$ and $\theta^{*} \nsupseteq s(\psi)$ must be true. Since $\theta^{*}=s(\delta) \cap s(\psi)$, then it is also true that $\theta^{*} \supseteq s(\delta) \cap s(\psi)$. Additionally, given that $\delta \nmid \psi$ implies $s(\delta) \nsubseteq s(\psi)$ (and $s(\psi) \nsubseteq s(\delta)$ ), then it also implies $s(\delta) \neq s(\psi)$. Thus, $\theta^{*}=s(\delta) \cap s(\psi) \neq s(\delta) \neq s(\psi)$, and we obtain $\theta^{*} \nsupseteq s(\delta)$ and $\theta^{*} \nsupseteq s(\psi)$, which finally shows that $\theta^{*} \supseteq_{\cap}(s(\delta), s(\psi))$. Then, for $\theta=\theta^{*}$, we have $(\delta \bowtie \psi)_{\| \theta^{*}} \equiv(\delta \bowtie \psi)_{\| \theta^{*}}$, as both relations have exactly the same schema and instance. This in turn implies that $(\delta \bowtie \psi)_{\| \theta^{*}} \sqsubseteq(\delta \bowtie \psi)_{\| \theta^{*}}$. So, $\mathrm{f}_{t}$ satisfies ( $\mathbf{T}^{-}$).

In order to prove that $\mathrm{f}_{t^{-}}$satisfies $\left(\mathbf{T}^{-}\right)$, let $D=(R, F), \delta$ and $\psi$ be as above. By the definition of the operator, for $\delta \nmid \psi$, we have $\mathrm{f}_{t^{-}}(D, \delta, \psi)=\left(R \backslash\{\delta\} \cup\left\{\left(\delta^{*} \bowtie \psi\right)_{\left.\| \theta^{*}\right\}}, F_{\| \delta}\right)\right.$, where $\delta^{*} \sqsubseteq \delta$ and $\theta^{*}=s(\delta) \cap s(\psi)$. Thus, $R_{\mathrm{f}_{t^{-}}(D, \delta, \psi)} \backslash R$ is equal to $\left(R \backslash\{\delta\} \cup\left\{\left(\delta^{*} \bowtie \psi\right)_{\left.\left.\| \theta^{*}\right\}\right) \backslash R}\right.\right.$, which can be simplified to $\left(\delta^{*} \bowtie \psi\right)_{\| \theta^{*}}$. For the same reasons as in the proof above, we can conclude that $\theta^{*} \supseteq_{\cap}(s(\delta), s(\psi))$ for $\delta \nmid \psi$. Therefore, for $\theta^{*}=\theta$, we must prove that
$\left(\delta^{*} \bowtie \psi\right)_{\| \theta^{*}} \sqsubseteq(\delta \bowtie \psi)_{\| \theta^{*}}$ is true. Considering that $\delta^{*} \sqsubseteq \delta$ implies $s\left(\delta^{*}\right)=s(\delta)$ and $i\left(\delta^{*}\right) \subseteq i(\delta)$, and that the operator $\bowtie$ is monotonic (Proposition 4.2.2 in [AHV95]), the expression $\left(\delta^{*} \bowtie \psi\right)_{\| \theta^{*}} \sqsubseteq(\delta \bowtie \psi)_{\| \theta^{*}}$ must be true.

Regarding the negative results, let us start by proving that $\mathrm{f}_{n t}$ does not satisfy $\left(\mathrm{T}^{-}\right)$. By the definition of the operator, we have $\mathrm{f}_{n t}(D, \delta, \psi)=(R \backslash\{\delta\}, F)$ for any $D=(R, F) \in \mathcal{D}_{\mathcal{A}}$, $\delta \in R$ and $\psi \in R$. Since $R_{\mathrm{f}_{n t}(D, \delta, \psi)} \backslash R=(R \backslash\{\delta\}) \backslash R=\emptyset$, then it is obvious that ( $\left.\mathbf{T}^{-}\right)$is not satisfied by $f_{n t}$.

Finally, for $\mathrm{f}_{t^{+}}$, consider a database $D=(R, F)$ such that $R=(\delta, \psi)$, where $\delta$ and $\psi$ correspond to the relations depicted by the tables below, and $F=\{A \rightarrow B ; B \rightarrow C\}$.


The result of $\mathrm{f}_{t^{+}}(D, \delta, \psi)$ is equal to the database $\left(\left(\psi, \phi^{*}\right), \emptyset\right)$, where $\phi^{*}$ corresponds to the transitive relation and is represented by the table below.

| $\phi^{*}$ |  |
| :---: | :---: |
| $A$ | $C$ |
| $\mathrm{a}_{1}$ | $\mathrm{c}_{1}$ |
| $\mathrm{a}_{1}$ | $\mathrm{c}_{2}$ |

Thus, $R_{\mathrm{f}_{t^{+}}(D, \delta, \psi)} \backslash R=\phi^{*}$. For $(\delta \bowtie \psi)_{\| \theta}$ to have the same schema as $\phi^{*}$, then $\theta=s(\delta) \cap s(\psi)=$ $\{B\}$ and, in that case, $(\delta \bowtie \psi)_{\| \mid B\}}$ corresponds to the relation shown in the next table.

| $(\delta \bowtie \psi)_{\{B\}}$ |  |
| :--- | :--- |
| $A$ | $C$ |
| $\mathrm{a}_{1}$ | $\mathrm{c}_{1}$ |

It is easy to see $i\left(\delta^{*}\right) \subseteq i\left(\delta \bowtie \psi_{\{B\}}\right)$ is not true. This implies that $\phi^{*} \sqsubseteq(\delta \bowtie \psi)_{\mid\{B\}}$ does not hold. Since there is no other relation in $R_{\mathrm{f}^{+}(D, \delta, \psi)} \backslash R$ besides $\phi^{*}$, we proved that $\mathrm{f}_{t^{+}}$does not satisfy ( $\mathbf{T}^{-}$).

Definition 5.10 (Transitivity with Addition $\left(\mathrm{T}^{+}\right)$). A relation forgetting operator f satisfies Transitivity with (possible) Addition if, for each $D=(R, F) \in \mathcal{D}_{\mathcal{A}}$ and $\delta, \psi \in R$ with $\delta \nmid \psi$, there exists $\phi \in R_{f(D, \delta, \psi)} \backslash R$ such that $(\delta \bowtie \psi)_{\| \theta} \sqsubseteq \phi$ for some $\theta \supseteq_{\cap}(s(\delta), s(\psi))$, if $\Pi_{\theta}(\psi)$ has at least two tuples.

Particularly for $\left(\mathbf{T}^{+}\right)$, note that we restrict the desirability of the transitive relation to the cases where the projection of $\theta$ in $\psi$ has at least two tuples, which goes in line with the argument made upon the introduction of $f_{t^{+}}$. In practice, this restriction does not weigh much importance in real-world scenarios, where we would already expect relations to have a large set of tuples.

Proposition 5.15. $\mathrm{f}_{t}$ and $\mathrm{f}_{t^{+}}$satisfy $\left(\mathrm{T}^{+}\right)$, but $\mathrm{f}_{n t}$ and $\mathrm{f}_{t^{-}}$do not.

Proof. To prove that $\mathrm{f}_{t}$ satisfies $\left(\mathbf{T}^{+}\right)$, let $D=(R, F) \in \mathcal{D}_{\mathcal{A}}, \delta \in R$ and $\psi \in R$. If $\delta \nmid \psi$ holds, then by the definition of the operator, we have $\mathrm{f}_{t}(D, \delta, \psi)=\left(R \backslash\{\delta\} \cup\left\{(\delta \bowtie \psi)_{\| \theta^{*}}\right\}, F_{\| \delta}\right)$, where $\theta^{*}=s(\delta) \cap s(\psi)$. As shown in the proof for Proposition 5.14, $R_{\mathrm{f}_{t}(D, \delta, \psi)} \backslash R=(\delta \bowtie \psi)_{\| \theta^{*}}$ and $\theta^{*} \supseteq_{\cap}(s(\delta), s(\psi))$ for $\delta \nmid \psi$. Therefore, we can assume that $\theta=\theta^{*}$. In that case, $(\delta \bowtie \psi)_{\| \theta^{*}} \equiv(\delta \bowtie \psi)_{\| \theta^{*}}$, as both relations have exactly the same schema and instance. This implies that $(\delta \bowtie \psi)_{\| \theta^{*}} \sqsubseteq(\delta \bowtie \psi)_{\| \theta^{*}}$, independently of the number of tuples in $\Pi_{\theta^{*}}(\psi)$. Thus, we proved that $\mathrm{f}_{t}$ satisfies $\left(\mathrm{T}^{+}\right)$.

In the case of $\mathrm{f}_{t^{+}}$, let $D=(R, F), \delta$ and $\psi$ be as above. Thus, if $\delta \nmid \psi$ holds, then by the definition of the operator, we have $\mathrm{f}_{t^{+}}(D, \delta, \psi)=\left(R \backslash\{\delta\} \cup\left\{\left(\delta \bowtie \psi^{\prime}\right)_{\| \theta^{*}}\right\}, F_{\| \delta}\right)$, where $\psi \sqsubseteq \psi^{\prime}$ and $\theta^{*}=s(\delta) \cap s(\psi)$, if there are at least two tuples in $\Pi_{\theta^{*}}(\psi)$ (i.e., $\exists t, t^{\prime} \in \Pi_{\theta}(\psi)$ s.t. $\left.t \neq t^{\prime}\right)$. Observe that this must be true because, if there are at least two tuples in $\Pi_{\theta^{*}}(\psi)$, then by the way that $\psi^{\prime}$ is defined (line 9 of Algorithm 6), for some $i$ in line 8 (potentially the last), the conditions in line 12 will not be satisfied for all tuples (since $r$, in line 5 , is equal to all combinations of values between $\Pi_{\theta^{*}}(\psi)$ and $\Pi_{s(\psi) \backslash \theta^{*}}(\psi)$ that are not in $\left.\psi\right)$. Therefore, $R_{\mathrm{f}_{t^{+}}(D, \delta, \psi)} \backslash R$ is equal to $\left(R \backslash\{\delta\} \cup\left\{\left(\delta \bowtie \psi^{\prime}\right)_{\| \theta^{*}}\right\}\right) \backslash R$, which in turn is equal to $\left(\delta \bowtie \psi^{\prime}\right)_{\| \theta^{*}}$. Furthermore, given that $\theta^{*}=s(\delta) \cap s(\psi) \supseteq_{\cap}(s(\delta), s(\psi))$ for $\delta \nmid \psi$ by the proof above, we an assume $\theta=\theta^{*}$. Thus, we must prove that $(\delta \bowtie \psi)_{\| \theta^{*}} \sqsubseteq\left(\delta \bowtie \psi^{\prime}\right)_{\| \theta^{*}}$. Since $\psi \sqsubseteq \psi^{\prime}$ is true by the definition of the operator and implies $i(\psi) \subseteq i\left(\psi^{\prime}\right)$, then taking into account that $\bowtie$ is monotonic (Proposition 4.2.2 in [AHV95]), the expression $(\delta \bowtie \psi)_{\| \theta^{*}} \sqsubseteq\left(\delta \bowtie \psi^{\prime}\right)_{\| \theta^{*}}$ must be true. This proves that $f_{t^{+}}$satisfies ( $T+$ ).

We now prove that $\mathrm{f}_{n t}$ does not satisfy $\left(\mathrm{T}^{+}\right)$. By the definition of the operator, we have $\mathrm{f}_{n t}(D, \delta, \psi)=(R \backslash\{\delta\}, F)$, for any $D=(R, F) \in \mathcal{D}_{\mathcal{A}}, \delta \in R$ and $\psi \in R$. Then, since $R_{\mathrm{f}_{n t}(D, \delta, \psi)} \backslash R=(R \backslash\{\delta\}) \backslash R=\emptyset$, it is straightforward that $\left(\mathbf{T}^{+}\right)$is not satisfied by $\mathrm{f}_{n t}$.

At last, to show that $\mathrm{f}_{t^{-}}$does not satisfy $\left(\mathbf{T}^{+}\right)$, let $D=(R, F), \delta$ and $\psi$ be as above. By the definition of the operator, for $\delta \nmid \psi$, we have $\mathrm{f}_{t^{-}}(D, \delta, \psi)=\left(R \backslash\{\delta\} \cup\left\{\left(\delta^{*} \bowtie \psi\right)_{\| \theta^{*}}\right\}, F_{\| \delta}\right)$, where $\delta^{*} \sqsubseteq \delta$ and $\theta^{*}=s(\delta) \cap s(\psi)$. Therefore, as was shown in the proof for Proposition 5.14, $R_{\mathrm{f}_{t^{-}}(D, \delta, \psi)} \backslash R=\left(\delta^{*} \bowtie \psi\right)_{\| \theta^{*}}$ and $\theta^{*} \supseteq_{\cap}(s(\delta), s(\psi))$ for $\delta \nmid \psi$. Now, since $\delta^{*} \sqsubseteq \delta$ implies $i\left(\delta^{*}\right) \subseteq i(\delta)$, then by Proposition 4.4 .2 in [AHV95], we have $\left(\delta^{*} \bowtie \psi\right)_{\| \theta^{*}} \sqsubseteq(\delta \bowtie \psi)_{\| \theta^{*}}$. So, even if $\theta=\theta^{*}$, we cannot satisfy the condition for $\left(\mathrm{T}^{+}\right)$. Considering that there are no other relations in $R_{\mathrm{f}_{t^{-}}(D, \delta, \psi)} \backslash R$ apart from $\left(\delta^{*} \bowtie \psi\right)_{\| \theta^{*}}$, we can conclude that $\mathrm{f}_{t^{-}}$does not obey ( $\mathrm{T}^{+}$).

To conclude the exposition and analysis of the characteristics that motivated our operators of forgetting, we turn our attention to perhaps one of the most crucial properties when it comes to forgetting relations.

As we have been mentioning throughout this dissertation, we are primarily interested in operators that do not allow intentional recovery of tuples which belonged to the relation that was forgotten. In Section 4.5, we elaborated on this intuition and stated that we can only ensure that no tuple in that relation can be inferred after the operation when, for
each such tuple, there is at least an alternative tuple (or set of tuples) that leads to exactly the same result.

More concretely, let us consider a database $D \in \mathcal{D}_{\mathcal{A}}$ and a relation to be forgotten $\delta \in R_{D}$. Then, for each tuple $t \in i(\delta)$, it must exist a relation $\delta^{\prime}$ with the same schema as $\delta$ such that $t \notin i\left(\delta^{\prime}\right)$ and the result of forgetting about $\delta^{\prime}$ in $D^{\prime}$, where $D^{\prime}$ is obtained by exchanging $\delta$ for $\delta^{\prime}$ in $R_{D}$, is equal to the result of forgetting about $\delta$ in $D$, both at the level of the relations as well as the $\mathrm{FDs}^{2}$. In this way, given the relation $\psi$ and the result of forgetting, even if we know the operator that was used to compute it, we cannot distinguish which relation $\delta$ or $\delta^{\prime}$ was in its input, and thus cannot conclude if $t$ was forgotten.

Any relation forgetting operator satisfying these conditions obeys No Recovery of Forgotten Tuples. Nevertheless, before presenting this property, we introduce notation to describe exchange of relations in a set. For that, let $R$ be a relation set over $\mathcal{A}$ that includes relation $r$, and $r^{\prime}$ any relation over $\mathcal{A}$. Then, the expression $R_{r \leftarrow r^{\prime}}$ is used as a shorthand for $R \backslash\{r\} \cup\left\{r^{\prime}\right\}$.

Definition 5.11 (No Recovery of Forgotten Tuples (NRT)). A relation forgetting operator f satisfies No Recovery of Forgotten Tuples if, for all $(R, F) \in \mathcal{D}_{\mathcal{A}}, \delta \in R, \psi \in \mathcal{R}_{\mathcal{A}} \cup\{\emptyset\}$ and for each $t \in i(\delta)$, there exists a relation $\delta^{\prime} \notin R$ such that $t \notin i\left(\delta^{\prime}\right)$ and $s\left(\delta^{\prime}\right)=s(\delta)$ for which $\mathrm{f}\left(\left(R_{\delta \leftarrow \delta^{\prime}}, F\right), \delta^{\prime}, \psi\right)=\mathrm{f}((R, F), \delta, \psi)$.

Before we go any further, note the importance of restricting the definition of (NRT) to the cases where $\delta$ belongs to $R$. If this was not done, then for the occasions where $\delta$ is not in $R$, we would be comparing the result of forgetting about this relation with the result of forgetting about $\delta^{\prime}$ in a database where it belongs to. Evidently, in such situations, we would expect the results not to be equal and therefore the property not to be satisfied.

Still regarding the definition of (NRT), we point out the fact that this property can be expressed in an alternative way. To that end, first notice that for each $\delta$, if we looked at all the alternative relations to $\delta$, in the sense that the result of forgetting about them is the same as $\delta$, and if there was a tuple common to all these alternatives, then we could conclude that this tuple had to be in the relation to be forgotten. Obviously, this would immediately imply recovery of forgotten tuples. To avoid this, there can be no tuple that belongs to all alternatives to $\delta$, i.e., the intersection of the instances of all these alternatives must be empty.

Proposition 5.16. Let f be a relation forgetting operator. Then, f satisfies (NRT) if and only if, for all $(R, F), \delta$ and $\psi$ as in Definition 5.11, and $\Delta=\left\{\delta^{\prime} \notin R \backslash\{\delta\} \mid s\left(\delta^{\prime}\right)=s(\delta)\right.$ and $\left.\mathrm{f}\left(\left(R_{\delta \leftarrow \delta^{\prime}}, F\right), \delta^{\prime}, \psi\right)=\mathrm{f}((R, F), \delta, \psi)\right\}$, the intersection of the instances of all relations in $\Delta$ is equal to the empty set, i.e., $\bigcap_{\delta^{\prime} \in \Delta^{\prime}} i\left(\delta^{\prime}\right)=\emptyset$.

[^22]Proof. We prove the implication in both directions. First, if (NRT) holds, then for each tuple in $\delta$, there is at least a relation $\delta^{\prime}$ satisfying the conditions imposed in $\Delta$ which does not have that tuple in its instance. Therefore, since $\delta$ is also in $\Delta$, the intersection of the instances of all relations in $\Delta$ must be equal to the empty set.

On the other hand, if the intersection of the instances of all relations in $\Delta$ is equal to the empty set, then for each tuple in the instance of any relation in $\Delta$, it must exist a different relation also in $\Delta$ that does not have that tuple in its instance. Since $\delta$ belongs to $\Delta$, then this is true for all tuples in $\delta$ and thus (NRT) must hold.

Even though Definition 5.11 expresses (NRT) in a "dynamic" way (i.e., with respect to each tuple in the instance of $\delta$ ), we just showed that it conveys precisely the same idea as the alternative characterisation given in Proposition 5.16.

Moreover, note that although the relevance of (NRT) was specifically brought up in the context of transitive forgetting, and that non-transitive operators do not introduce any (transitive) information in the database, (NT) does not imply (NRT) as, perhaps, could be assumed. The reason for this is that the result of the operators that satisfy (NT) does not necessarily have to be independent of $\delta$.

Proposition 5.17. (NT) does not imply (NRT).
Proof. Consider a relation forgetting operator f such that, for any $D=(R, F) \in \mathcal{D}_{\mathcal{A}}, \delta \in \mathcal{D}_{\mathcal{A}}$ and $\psi \in \mathcal{D}_{\mathcal{A}}$, we have $\mathrm{f}(D, \delta, \psi)=(R \backslash\{\delta\}, F)$ if a specific tuple $t$ belongs to the instance of $\delta$ and $\mathrm{f}(D, \delta, \psi)=\left(R \backslash\{\delta\}, F^{+}\right)$otherwise. To be more specific, let us assume that $t=\left(\mathrm{a}_{1}, \mathrm{~b}_{1}\right)$.

In order to prove that the operator is well defined we can resort to the proof for Proposition 5.1, which would be very similar to this one. Thus, we start by proving instead that f satisfies (NT). To that end, we must show that $\mathrm{f}(D, \delta, \psi) \equiv_{\delta} D$ for any database in the output, i.e., $(R \backslash\{\delta\}, F) \equiv_{\delta} D$ and $\left(R \backslash\{\delta\}, F^{+}\right) \equiv_{\delta} D$. Regarding the first database, note that the $\delta$-equivalence can be written as $(R \backslash\{\delta\}, F) \equiv_{\delta}(R, F)$, which by definition implies $(R \backslash\{\delta\}, F) \equiv(R \backslash\{\delta\}, F)$. This, in turn, is true by Proposition 3.2. Regarding the second database, we must prove that $\left(R \backslash\{\delta\}, F^{+}\right) \equiv_{\delta}(R, F)$, which is equivalent to $\left(R \backslash\{\delta\}, F^{+}\right) \equiv(R \backslash\{\delta\}, F)$, is true. Now, since $(R \backslash\{\delta\}, F) \equiv(R \backslash\{\delta\}, F)$ is true, then it implies $(R \backslash\{\delta\}, F) \leqslant(R \backslash\{\delta\}, F)$, and since $F^{+}=F^{+}$is also true, then by Proposition 3.3 we have $(R \backslash\{\delta\}, F) \leqslant\left(R \backslash\{\delta\}, F^{+}\right)$and $\left(R \backslash\{\delta\}, F^{+}\right) \leqslant(R \backslash\{\delta\}, F)$. This in turn implies $\left(R \backslash\{\delta\}, F^{+}\right) \equiv$ $(R \backslash\{\delta\}, F)$, proving that f satisfies (NT).

Now, consider a database $D=((\delta, \psi),\{A \rightarrow B ; B \rightarrow C\})$, where $\delta$ and $\psi$ correspond to the relations shown below.

| $\delta$ |  |  | $\psi$ |  |
| :---: | :---: | :---: | :---: | :---: |
|  |  | $B$ | $B$ |  |
|  |  | $C$ |  |  |

It is easy to see that there is no relation $\delta^{\prime}$ with the same schema as $\delta$ and without the tuple ( $\mathrm{a}_{1}, \mathrm{~b}_{1}$ ) that would give the same result as $\mathrm{f}(D, \delta, \psi)=\left((\psi), F_{D}\right)$ when exchanged
in $D$ with $\delta$, since for any $\delta^{\prime}$ without ( $\mathrm{a}_{1}, \mathrm{~b}_{1}$ ), we would have $A \rightarrow C \in F_{\mathrm{f}\left(D^{\prime}, \delta^{\prime}, \psi\right)}$, where $D^{\prime}=\left(\left(R_{D}\right)_{\delta \leftarrow \delta^{\prime}}, F_{D}\right)$. Yet, $A \rightarrow C \notin F_{\mathrm{f}(D, \delta, \psi)}$. Therefore, (NT) does not imply (NRT).

However, $\mathrm{f}_{n t}$ in particular still obeys (NRT), given that it imposes an equality between the original and resulting database that is, by construction, independent of $\delta$. Most importantly, although the transitive operator $\mathrm{f}_{t}$ does not satisfy (NRT), the operators $\mathrm{f}_{t^{-}}$ and $f_{t+}$ satisfy it, even though the latter only does for a restricted domain.

Accordingly, in the next proposition, we show that $f_{n t}$ and $f_{t^{-}}$satisfy (NRT) for any $(R, F), \delta$ and $\psi$ in the domains defined in the property, and that $\mathrm{f}_{t}$ and $\mathrm{f}_{t^{+}}$do not. Afterwards, we prove that if no FDs with attributes of both $s(\delta) \backslash \theta$ and $\theta$ are in $F^{+}$, where $\theta$ is the intersection of the schemas of $\delta$ and $\psi$, then $\mathrm{f}_{t^{+}}$also satisfies (NRT).

Proposition 5.18. $\mathrm{f}_{n t}$ and $\mathrm{f}_{t^{-}}$satisfy (NRT), but $\mathrm{f}_{t}$ and $\mathrm{f}_{t^{+}}$do not.
Proof. To prove that $\mathrm{f}_{n t}$ satisfies (NRT) we will use the alternative characterisation of this property given by Proposition 5.16. Let $(R, F) \in \mathcal{D}_{\mathcal{A}}, \delta \in R, \psi \in \mathcal{R}_{\mathcal{A}} \cup\{\emptyset\}$ and $\Delta=\left\{\delta^{\prime} \notin\right.$ $\left.R \backslash\{\delta\} \mid s\left(\delta^{\prime}\right)=s(\delta) \wedge \mathrm{f}_{n t}\left(\left(R_{\delta \leftarrow \delta^{\prime}}, F\right), \delta^{\prime}, \psi\right)=\mathrm{f}_{n t}((R, F), \delta, \psi)\right\}$. By definition of the operator, $\mathrm{f}_{n t}((R, F), \delta, \psi)=(R \backslash\{\delta\}, F)$. Furthermore, for any $\delta^{\prime} \in \mathcal{R}_{\mathcal{A}}$, we have $\mathrm{f}_{n t}\left(\left(R_{\delta \leftarrow \delta^{\prime}}, F\right), \delta^{\prime}, \psi\right)=$ $\mathrm{f}_{n t}\left(\left(R \backslash\{\delta\} \cup\left\{\delta^{\prime}\right\}, F\right), \delta^{\prime}, \psi\right)$, which again by definition is equal to $\left(R \backslash\left\{\delta, \delta^{\prime}\right\}, F\right)$. Therefore, $\Delta$ can be simplified to $\left\{\delta^{\prime} \notin R \backslash\{\delta\} \mid s\left(\delta^{\prime}\right)=s(\delta) \wedge\left(R \backslash\left\{\delta, \delta^{\prime}\right\}, F\right)=(R \backslash\{\delta\}, F)\right\}$. Now, if we consider any relation $\delta^{\prime} \in \mathcal{R}_{\mathcal{A}} \backslash(R \backslash\{\delta\})$ such that $s\left(\delta^{\prime}\right)=s(\delta)$, then it is straightforward to see that the equality $\left(R \backslash\left\{\delta, \delta^{\prime}\right\}, F\right)=(R \backslash\{\delta\}, F)$ is true because, taken into account that $\delta^{\prime} \notin R \backslash\{\delta\}$, then $R \backslash\{\delta\}=(R \backslash\{\delta\}) \backslash\left\{\delta^{\prime}\right\}=R \backslash\left\{\delta, \delta^{\prime}\right\}$. This means that $\delta^{\prime} \in \Delta$, independently of its instance. As such, for $i\left(\delta^{\prime}\right)=\emptyset$, the intersection of the instances of all relations in $\Delta$ can only be the empty set. So, $\mathrm{f}_{n t}$ satisfies (NRT).

For the operator $\mathrm{f}_{t^{-}}$, we must show that it satisfies (NRT) both when the conditions for transitivity are satisfied and when they are not. Let $(R, F), \delta$ and $\psi$ be as above. Then, if the conditions for transitivity are not satisfied, i.e., $\psi \notin R$ or $\delta \nmid \psi$ does not hold (note that by the definition of (NRT) $\delta$ must be in $R$ ), we have $\mathrm{f}_{t^{-}}((R, F), \delta, \psi)=(R \backslash\{\delta\}, F)$. Similarly to the proof for the operator $\mathfrak{f}_{n t}$, for any $\delta^{\prime} \in \mathcal{R}_{\mathcal{A}}$ such that $s(\delta)=s\left(\delta^{\prime}\right)$, we have $\mathrm{f}_{t^{-}}\left(\left(R_{\delta \leftarrow \delta^{\prime}}, F\right), \delta^{\prime}, \psi\right)=\left(R \backslash\left\{\delta, \delta^{\prime}\right\}, F\right)$ since the conditions for transitivity are also not satisfied for $\delta^{\prime}$ as its schema is equal to the one of $\delta$ (thus, either $\delta^{\prime} \nmid \psi$ is not true if $\delta \nmid \psi$ is not or $\psi \notin R$ in the first place). Now, assuming the characterisation of (NRT) given by Proposition 5.16, if we consider any $\delta^{\prime}$ that does not belong to $R \backslash\{\delta\}$, then, as we showed above, it is always true that $\mathrm{f}_{t^{-}}\left(\left(R_{\delta \leftarrow \delta^{\prime}}, F\right), \delta^{\prime}, \psi\right)$ is equal to $\mathrm{f}_{t^{-}}((R, F), \delta, \psi)$, since $\left(R \backslash\left\{\delta, \delta^{\prime}\right\}, F\right)=(R \backslash\{\delta\}, F)$. Therefore, for $\Delta$ as in the proposition, we have $\delta^{\prime} \in \Delta$, regardless of the instance of $\delta^{\prime}$. Using the same strategy as the proof for $\mathrm{f}_{n t}$, if $i\left(\delta^{\prime}\right)=\emptyset$, then the intersection of the instances of all relations in $\Delta$ must be the empty set as well.

On the other hand, in case the conditions for transitivity are satisfied, i.e., $\psi \in R$ and $\delta \nmid \psi$, then, using the definition of (NRT), we must show that for each $t \in i(\delta)$, there exists a relation $\delta^{\prime} \notin R$ such $t \notin \delta^{\prime}, s(\delta)=s\left(\delta^{\prime}\right)$ and $\mathrm{f}_{t^{-}}\left(\left(R_{\delta \leftarrow \delta^{\prime}}, F\right), \delta^{\prime}, \psi\right)=\mathrm{f}_{t^{-}}((R, F), \delta, \psi)$. To that end, observe that, by the definition of the operator in the case of transitivity, we have
$\mathrm{f}_{t^{-}}((R, F), \delta, \psi)=\left(R \backslash\{\delta\} \cup\left\{\left(\delta^{*} \bowtie \psi\right)_{\| \theta\}}, F_{\| \delta}\right)\right.$, where $\theta=s(\delta) \cap s(\psi)$, and therefore we want the result of $\mathrm{f}_{t^{-}}\left(\left(R_{\delta \leftarrow \delta^{\prime}}, F\right), \delta^{\prime}, \psi\right)$ to be the same database. Thus, in order to show that for each $t \in i(\delta)$, there is always a relation $\delta^{\prime}$ satisfying all these requirements, we split the proof in two parts:
(1) we assume that $i\left(\delta^{\prime}\right)=i\left(\delta^{*}\right)$ if $t \notin i\left(\delta^{*}\right)$;
(2) we assume that $i\left(\delta^{\prime}\right)=i\left(\delta^{*}\right) \backslash\{t\} \cup\left\{t^{*}\right\}$, where $t^{*}$ corresponds to the alternative to $t$ (line 9 of Algorithm 5), if $t \in i\left(\delta^{*}\right)$.

For both cases we must show that
(a) $\left(R_{\delta \leftarrow \delta^{\prime}}, F\right)$, where $R_{\delta \leftarrow \delta^{\prime}}$ corresponds to $R \backslash\{\delta\} \cup\left\{\delta^{\prime}\right\}$, is a database, i.e., the relation set satisfies $F^{+}$; and
(b) $\mathrm{f}_{t^{-}}\left(\left(R_{\delta \leftarrow \delta^{\prime}}, F_{\| \delta}\right), \delta^{\prime}, \psi\right)=\left(R \backslash\{\delta\} \cup\left\{\left(\delta^{*} \bowtie \psi\right)_{\| \theta\}}\right\}, F\right)$.

Let us start with case (1). To prove (a), it suffices to show that $\delta^{\prime}$ satisfies the FDs in $F_{\delta^{\prime}}$, as the other relations in the set are the same as those in the database received as input for the operator. Since $s(\delta)=s\left(\delta^{\prime}\right)$, then $F_{\delta^{\prime}}=F_{\delta}$. Thus, given that $\delta$ satisfies $F_{\delta}$ (otherwise $(R, F)$ would not be a valid input for the operator, given that $\delta \in R$ ), then any subset of its instance also does. In particular, $i\left(\delta^{\prime}\right)$ satisfies $F_{\delta}=F_{\delta^{\prime}}$, considering that $i\left(\delta^{\prime}\right)=i\left(\delta^{*}\right)$ and $i\left(\delta^{*}\right)$ is a subset of $i(\delta)$ by the construction of the operator (line 13).

Now, to show that (b) also holds, observe first that $\delta^{*}$ is obtained by applying a test to each tuple in $i(\delta)$ (line 6 of Algorithm 5). Thus, since $i\left(\delta^{*}\right) \subseteq i(\delta)$ by construction, we must prove that all tuples in $i\left(\delta^{*}\right)$ also pass the same test and therefore are in $\delta^{*}$ of $\delta^{*}$, i.e. $\delta^{* *}$ (note that the conditions for transitivity are also satisfied by $\delta^{*}$ since it has the same schema as $\delta$ ). To that end, let us assume that there is a tuple $t^{*} \in i\left(\delta^{*}\right)$ that does not pass the test. In that case, it does not exist a tuple $t^{* *} \in \Pi_{s\left(\delta^{*}\right) \backslash \theta}\left(\delta^{*}\right) \times \Pi_{\theta}(\psi)$ such that
(i) $t^{*} \neq t^{* *}$ and
(ii) $t^{*}\left[s\left(\delta^{*}\right) \backslash \theta\right]=t^{* *}\left[s\left(\delta^{*}\right) \backslash \theta\right]$ and
(iii) $\Pi_{s(\psi) \backslash \theta}\left(\sigma_{\theta=t^{*}[\theta]}(\psi)\right)=\prod_{s(\psi) \backslash \theta}\left(\sigma_{\theta=t^{* *}[\theta]}(\psi)\right)$ and
(iv) for all $X \rightarrow Y \in F^{+}$such that $X \cup Y \nsubseteq \theta$ and $\exists t^{\prime} \in i\left(\delta^{*}\right) \backslash\left\{t^{*}\right\}$ s.t. $t^{\prime}[X \backslash \theta]=t[X \backslash \theta]$ then $t^{*}[X \cup Y]=t^{* *}[X \cup Y]$.

However, for $t^{*}$ to be in $i\left(\delta^{*}\right)$, then it must exist a tuple $t^{\dagger} \in \Pi_{s\left(\delta^{*}\right) \backslash \theta}\left(\delta^{*}\right) \times \Pi_{\theta}(\psi)$ that satisfies at least the conditions (i), (ii) and (iii) for $t^{*}$, as they do not depend on $i\left(\delta^{*}\right)$ (note that the condition (ii) guarantees that $t^{\dagger}$ is in the correct domain, i.e. $t^{\dagger} \in \Pi_{s\left(\delta^{*}\right) \backslash \theta}\left(\delta^{*}\right) \times \Pi_{\theta}(\psi)$, since $\psi$ is fixed). In addition, since $i\left(\delta^{*}\right) \subseteq i(\delta)$, then $i\left(\delta^{*}\right) \backslash\left\{t^{*}\right\} \subseteq i(\delta) \backslash\left\{t^{*}\right\}$. Thus, if the antecedent of the implication for the condition (iv) is true for $t^{*}$ in $\delta^{*}$, then it is also true for $t^{*}$ in $\delta$ (considering that if $t^{\prime}$ belongs to $i\left(\delta^{*}\right) \backslash\left\{t^{*}\right\}$ then it also belongs to $i(\delta) \backslash\left\{t^{*}\right\}$ ). Therefore, $t^{*}[X \cup Y]=t^{\dagger}[X \cup Y]$ must be true, otherwise $t^{*}$ would not be in $\delta^{*}$. Thus, $t^{*}$ must pass the
test. As such, since there is no tuple in $\delta^{*}$ that does not pass the test, we have $\delta^{*}=\delta^{* *}$, and consequently, for $i\left(\delta^{\prime}\right)=i\left(\delta^{*}\right)$, we have $\mathrm{f}_{t^{-}}\left(\left(R_{\delta \leftarrow \delta^{\prime}}, F\right), \delta^{\prime}, \psi\right)=\left(R \backslash\{\delta\} \cup\left\{\left(\delta^{* *} \bowtie \psi\right)_{\| \theta}\right\}, F_{\| \delta}\right)=$ $\left(R \backslash\{\delta\} \cup\left\{\left(\delta^{*} \bowtie \psi\right)_{\| \theta}\right\}, F_{\| \delta}\right)$, which proves (b).

Regarding case (2), to prove (a), we just need to show that adding $t^{*}$ to $i\left(\delta^{*}\right) \backslash\{t\}$ does not infringe any FD, since we already proved for (1) that $i\left(\delta^{*}\right)$ satisfies $F^{+}$(and therefore any subset of it also satisfies). To that end, let us assume that it infringes. Then, it exists a functional dependency $X \rightarrow Y \in F^{+}$such that, for some $t^{\dagger} \in i\left(\delta^{*}\right)$, we have $t^{*}[X]=t^{\dagger}[X]$ and $t^{*}[Y] \neq t^{\dagger}[Y]$ (note that if it failed for another tuple other than $t^{*}$, then $i\left(\delta^{*}\right)$ would not satisfy $\left.F^{+}\right)$. However, since $t \in i\left(\delta^{*}\right)$ and $t^{*}$ is the alternative to $t$, then the condition in line 12 of Algorithm 5 must hold for any FD in $F^{+}$. In particular, it must hold for $X \rightarrow Y$. Thus, given that the condition holds, the implication must be true. As such, if we rewrite the implication into a disjuction, then for $X \rightarrow Y$, at least one of the following statements must be true.
(I) $X \cup Y \subseteq \theta$;
(II) $\nexists t^{\prime} \in i(\delta) \backslash\{t\}$ s.t. $t^{\prime}[X \backslash \theta]=t[X \backslash \theta]$;
(III) $t[X \cup Y]=t^{*}[X \cup Y]$.

Regarding (I), it cannot be true because if $X \rightarrow Y$ fails for $i\left(\delta^{*}\right) \backslash\{t\} \cup t^{*}$, then it would also fail for $\psi$, considering that $\theta \subseteq s(\psi), t^{*}[\theta] \in \Pi_{\theta}(\psi)$ and $t^{\dagger}[\theta] \in \Pi_{\theta}(\psi)$ (note that if $t^{\dagger}[\theta] \notin \Pi_{\theta}$, then $t^{\dagger}$ could not be in $i\left(\delta^{*}\right)$, as it would have failed the test of line 7 ). But since $\psi \in R$ and $R$ satisfies $F^{+}$, then $\psi$ also satisfies $F^{+}$and thus $X \rightarrow Y$ cannot fail in $\psi$. Before analysing (II), recall that, since $t^{*}$ is an alternative to $t$, then $t[s(\delta) \backslash \theta]=t^{*}[s(\delta) \backslash \theta]$ (because of line 10), which implies $t[X \backslash \theta]=t^{*}[X \backslash \theta]$. Thus, we can rewrite (II) as $\exists t^{\prime} \in i(\delta) \backslash\{t\}$ s.t. $t^{\prime}[X \backslash \theta]=t^{*}[X \backslash \theta]$. Now, it is obvious that this cannot be true, considering that if it was, then $t^{\dagger}[X \backslash \theta]$ and $t^{*}[X \backslash \theta]$ could not be equal, and therefore $t^{\dagger}[X]=t^{*}[X]$ would be false and $X \rightarrow Y$ not fail. Finally, (III) is also false, otherwise $X \rightarrow Y$ would fail in $i\left(\delta^{*}\right)$, which was already proved above that it does not. Since none of the statements is true, then $X \rightarrow Y$ does not satisfy the condition in line 12 , which means we derived a contradiction. Thus, we can conclude that no such functional dependency $X \rightarrow Y$ can exist in $F^{+}$, which implies that (a) is true.

To prove that (b) also holds for case (2), then we just need to show that if $i\left(\delta^{\prime}\right)=$ $i\left(\delta^{*}\right) \backslash\{t\} \cup\left\{t^{*}\right\}$, the result of $\left(\delta^{\prime} \bowtie \psi\right)_{\| \theta}$ is equal to $\left(\delta^{*} \bowtie \psi\right)_{\| \theta}$ (recall that $\delta^{\prime} \notin R$ and, since $s\left(\delta^{\prime}\right)=s\left(\delta^{\prime}\right)$ the conditions for transitivity are satisfied, otherwise they would not be for $\left.\delta\right)$. To that end, we start by proving that all tuples in $i\left(\delta^{\prime}\right)$ pass the test for $\left(\delta^{\prime}\right)^{*}$ (line 6). Since the tuples in $i\left(\delta^{*}\right)$ were already proved to pass this test and $t[s(\delta) \backslash \theta]=t^{*}[s(\delta) \backslash \theta]$ is true, given that $t^{*}$ is an alternative to $t$, then if we replace $t$ with $t^{*}$ in $i\left(\delta^{*}\right)$, all tuples would still pass the test, as the conditions in lines 9,10 and 11 are independent of $i\left(\delta^{*}\right)$ and, for the condition in line 12 , we would have $t^{*}$ in $i\left(\delta^{*}\right)$ such that $t[X \backslash \theta]=t^{*}[X \backslash \theta]$, making the condition exactly the same. Therefore, we can conclude that all tuples in $i\left(\delta^{\prime}\right)$ also pass the test. Thus, since $t$ and $t^{*}$ have exactly the same values for $s\left(\delta^{\prime}\right) \backslash \theta$ and their values for
the attributes in $\theta$ participate exactly with the same values in $\psi$ for the attributes $s(\psi) \backslash \theta$, then taking into account that $i\left(\delta^{\prime}\right)=i\left(\delta^{*}\right) \backslash\{t\} \cup\left\{t^{*}\right\}$ and the schema of $\left(\delta^{\prime} \bowtie \psi\right)_{\| \theta}$ has the attributes $s(\delta) \backslash \theta$ and $s(\psi) \backslash \theta$, the result of $\left(\delta^{\prime} \bowtie \psi\right)_{\| \theta}$ is equal to $\left(\delta^{*} \bowtie \psi\right)_{\| \theta}$. At last, this proves that $\mathrm{f}_{t^{-}}$satisfies (NRT).

We now show the negative results. Starting with $\mathrm{f}_{t}$, to prove that it does not satisfy (NRT), consider the database $D=((\delta, \psi),\{A \rightarrow B ; B \rightarrow C\})$, where $\delta$ and $\psi$ correspond to the relations shown below.


The result of $\mathrm{f}_{t}(D, \delta, \psi)$ would be the database $((\psi, \phi), \emptyset)$ where $\phi$ corresponds to the transitive relation represented by the next table.


It is obvious that in this case, it does not exist a relation $\delta^{\prime}$ with the same schema as $\delta$ and without the tuple $\left(\mathrm{a}_{1}, \mathrm{~b}_{1}\right)$ that would give the same result as $\mathrm{f}(D, \delta, \psi)=\left((\psi), F_{D}\right)$ when replaced in $D$ over $\delta$, since for the transitive relation to have the tuple ( $\mathrm{a}_{1}, \mathrm{c}_{1}$ ), then $\delta^{\prime}$ would need to have the tuple ( $a_{1}, \mathrm{~b}_{1}$ ), as it is the only tuple that can be joined with $\psi$ to achieve the same result as in $\phi$. Hence, $\mathrm{f}_{t}$ does not obey (NRT).

Finally, to prove that $\mathrm{f}_{t^{+}}$does not satisfy (NRT), consider the database $D B=(R, F)$ in Example 4.5 with $R=(\delta, \psi)$ and $F=\{B \rightarrow A ; D \rightarrow E\}$.

| $\delta$ |  |  |  |  |  | $\psi$ |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| $A$ $B$ $C$ |  | $B$ | $C$ | $D$ |  |  |  |  |
|  | $\mathrm{a}_{1}$ | $\mathrm{~b}_{1}$ | $\mathrm{c}_{1}$ |  | $\mathrm{~b}_{1}$ | $\mathrm{c}_{1}$ |  |  |
| $\mathrm{a}_{2}$ | $\mathrm{~b}_{2}$ | $\mathrm{c}_{3}$ |  |  |  |  |  |  |
|  |  |  | $\mathrm{~b}_{1}$ | $\mathrm{c}_{1}$ | $\mathrm{~d}_{2}$ |  |  |  |
|  |  | $\mathrm{~b}_{1}$ | $\mathrm{c}_{2}$ | $\mathrm{~d}_{1}$ |  |  |  |  |
| $\mathrm{~b}_{1}$ | $\mathrm{c}_{2}$ | $\mathrm{~d}_{2}$ |  |  |  |  |  |  |
|  |  |  | $\mathrm{~b}_{1}$ | $\mathrm{c}_{3}$ | $\mathrm{~d}_{3}$ |  |  |  |
| $\mathrm{~b}_{2}$ | $\mathrm{c}_{3}$ | $\mathrm{~d}_{3}$ |  |  |  |  |  |  |

Since $\delta \in R, \psi \in R, \delta \nmid \psi$ and $\psi^{\prime}$ with $i\left(\psi^{\prime}\right)=i(\psi)$ is valid for transitivity, the result of forgetting about $\delta$ with respect to $\psi$ in $D B$ is equal to the database $\left((\psi, \phi),\{D \rightarrow E\}^{+}\right)$, where $\phi$ corresponds to the transitive relation represented by the next table.

| $\phi$ |  |
| :---: | :---: |
| $A$ | $D$ |
| $\mathrm{a}_{1}$ | $\mathrm{~d}_{1}$ |
| $\mathrm{a}_{1}$ | $\mathrm{~d}_{2}$ |
| $\mathrm{a}_{2}$ | $\mathrm{~d}_{3}$ |

In this case, there is no relation $\delta^{\prime}$ such that $s\left(\delta^{\prime}\right)=s(\delta)$ and $t=\left(\mathrm{a}_{2}, \mathrm{~b}_{2}, \mathrm{c}_{3}\right) \notin i\left(\delta^{\prime}\right)$ for which $\mathrm{f}_{t^{+}}\left(\left(R_{\delta \leftarrow \delta^{\prime}}, F\right), \delta^{\prime}, \psi\right)$ outputs that exact transitive relation, since for $\phi$ to have the tuple ( $\mathrm{a}_{2}, \mathrm{~d}_{3}$ ), then $i\left(\delta^{\prime}\right)$ would need to have the tuple ( $\mathrm{a}_{2}, \mathrm{~b}_{1}, \mathrm{c}_{3}$ ), which would not be possible because of the functional dependency $A \rightarrow B$, as either $\left(a_{1}, b_{1}, c_{1}\right)$ or $\left(a_{1}, b_{1}, c_{2}\right)$ must be in $i\left(\delta^{\prime}\right)$ for $\left(\mathrm{a}_{1}, \mathrm{~d}_{1}\right)$ and $\left(\mathrm{a}_{1}, \mathrm{~d}_{2}\right)$ to be in $\phi$. Hence, $\mathrm{f}_{t^{+}}$does not satisfy (NRT), which finishes the proof for the proposition.

Proposition 5.19. If no FDs with attributes of both $s(\delta) \backslash \theta$ and $\theta$ are in $F^{+}$, where $\theta=$ $s(\delta) \cap s(\psi)$, then $\mathrm{f}_{t^{+}}$satisfies (NRT).

Proof. We divide the proof in two parts. In the first, we show that $\mathrm{f}_{t}$ satisfies (NRT) in case the conditions for transitivity are not satisfied. Subsequently, we do it for when they are. Let $(R, F), \delta$ and $\psi$ be as in the definition of the property.

If the conditions for transitivity are not satisfied, i.e., either $\psi \notin R$ or $\delta \nmid \psi$ does not hold (recall that $\delta$ must be in $R$ by de definition of (NRT)), we have $\mathrm{f}_{t^{+}}((R, F), \delta, \psi)=(R \backslash\{\delta\}, F)$. And thus, similarly to the proof for $\mathrm{f}_{n t}$ in Proposition 5.18 , for any $\delta^{\prime} \in \mathcal{R}_{\mathcal{A}}$ such that $s\left(\delta^{\prime}\right)=$ $s(\delta)$, we have $\mathrm{f}_{t^{+}}\left(\left(R_{\delta \leftarrow \delta^{\prime}}, F\right), \delta^{\prime}, \psi\right)=\left(R \backslash\left\{\delta, \delta^{\prime}\right\}, F\right)$ since that the conditions for transitivity are also not satisfied for $\delta^{\prime}$ as its schema is equal to the one of $\delta$ (thus, either $\delta^{\prime} \nmid \psi$ is not true if $\delta \nmid \psi$ is not or $\psi \notin R$ in the first place). Assuming the characterisation of (NRT) given by Proposition 5.16, if we consider any $\delta^{\prime}$ that does not belong to $R \backslash\{\delta\}$, then it is always true that $\mathrm{f}_{t^{+}}\left(\left(R_{\delta \leftarrow \delta^{\prime}}, F\right), \delta^{\prime}, \psi\right)$ is equal to $\mathrm{f}_{t^{+}}((R, F), \delta, \psi)$, since $\left(R \backslash\left\{\delta, \delta^{\prime}\right\}, F\right)=(R \backslash\{\delta\}, F)$. Therefore, for $\Delta=\left\{\delta^{\prime} \notin R \backslash\{\delta\} \mid s\left(\delta^{\prime}\right)=s(\delta)\right.$ and $\left.\mathrm{f}_{t^{+}}\left(\left(R_{\delta \leftarrow \delta^{\prime}}, F\right), \delta^{\prime}, \psi\right)=\mathrm{f}_{t^{+}}((R, F), \delta, \psi)\right\}$, we have $\delta^{\prime} \in \Delta$, regardless of the instance of $\delta^{\prime}$. Using the same strategy as the proof for $\mathrm{f}_{n t}$, if $i\left(\delta^{\prime}\right)=\emptyset$, then the intersection of the instances of all relations in $\Delta$ must be the empty set as well.

Now, if the conditions for transitivity are satisfied, there are two alternative databases in the result of the operator. The first one, which corresponds to the cases where $\psi^{\prime}$ is not valid for all $i \in$ ordered_pset (line 8 of Algorithm 6), is equal to $(R \backslash\{\delta\}, F)$. Since in these cases the result is independent of the instance of $\delta$, then even if $i(\delta)=\emptyset$, we would have the same database. Thus, for a relation $\delta^{\prime} \notin R$ such that $s\left(\delta^{\prime}\right)=s(\delta)$ and $i\left(\delta^{\prime}\right)=\emptyset$, the result of $\mathrm{f}_{t^{+}}\left(\left(R_{\delta \leftarrow \delta^{\prime}}, F\right), \delta^{\prime}, \psi\right)$ would be equal to $\left(R \backslash\left\{\delta, \delta^{\prime}\right\}, F\right)$, which in turn is equal to $(R \backslash\{\delta\}, F)$ (note that if $\delta \nmid \psi$ holds then $\delta^{\prime} \nmid \psi$ also holds, and that $\psi$ is fixed, thus if $\psi^{\prime}$ is not valid for $R$, it is also not valid for $R_{\delta \leftarrow \delta^{\prime}}$ ). Given that $\delta^{\prime}$ does not have any tuple, then it satisfies the conditions in the definition of the property for any $\delta$.

In all the remaining cases that there is transitivity, we have $\left.\mathrm{f}_{t^{+}}((R, F), \delta, \psi)\right\}=(R \backslash\{\delta\} \cup$ $\left.\left\{\left(\delta \bowtie \psi^{\prime}\right)_{\| \theta\}}\right\}, F_{\| \delta}\right)$, where $\psi \sqsubseteq \psi^{\prime} \sqsubseteq\left(\Pi_{\theta}(\psi) \times \Pi_{s(\psi) \backslash \theta}(\psi)\right)$ and $\theta=s(\delta) \cap s(\psi)$. Therefore,
we must prove that, for such cases, for any $t \in i(\delta)$, there exists a relation $\delta^{\prime}$ such that $t \notin \delta^{\prime}, s(\delta)=s\left(\delta^{\prime}\right)$ and $\left.\mathrm{f}_{t^{+}}\left(\left(R_{\delta \leftarrow \delta^{\prime}}, F\right), \delta^{\prime}, \psi\right)=\mathrm{f}_{t^{+}}((R, F), \delta, \psi)\right\}$. Thus, since the schemas of $\delta$ and $\delta^{\prime}$ are the same and $\psi$ is fixed, the result of $\mathrm{f}_{t^{+}}\left(\left(R_{\delta \leftarrow \delta^{\prime}}, F_{\| \delta}\right), \delta^{\prime}, \psi\right)$ is equal to $\left(R \backslash\left\{\delta, \delta^{\prime}\right\} \cup\left\{\left(\delta^{\prime} \bowtie \psi^{\prime}\right)_{\| \theta}\right\}, F_{\| \delta}\right)$. Taking into account that $\delta^{\prime} \notin R$, we have $R \backslash\left\{\delta, \delta^{\prime}\right\}=R \backslash\{\delta\}$, which implies that we just need to prove the equality $\left(\delta^{\prime} \bowtie \psi^{\prime}\right)_{\| \theta}=\left(\delta \bowtie \psi^{\prime}\right)_{\| \theta}$. To that end, note that the schema of the relation resulting from both operations has the attributes $s\left(\delta^{\prime}\right) \backslash \theta=s(\delta) \backslash \theta$ and $s\left(\psi^{\prime}\right) \backslash \theta$. Therefore, let us assume that $i\left(\delta^{\prime}\right)=i(\delta) \backslash\{t\} \cup T$. Then, for the equality to be true, $t\left[s\left(\delta^{\prime}\right) \backslash \theta\right]$ and $t^{\prime}\left[s\left(\delta^{\prime}\right) \backslash \theta\right]$ must be equal for all $t^{\prime} \in T, t[\theta]$ and $t^{\prime}[\theta]$ be different for all $t^{\prime} \in T$, so that $t \notin T$, and $t[\theta]$ and $T[\theta]$, where $T[\theta]=\left\{t^{\prime}[\theta] \mid t^{\prime} \in T\right\}$, must participate with exactly the same values of $s\left(\psi^{\prime}\right) \backslash \theta$ in $\psi^{\prime}$. Considering that $\psi^{\prime}$ is valid (otherwise the resulting database would not have the transitive relation), then it is guaranteed that for each tuple $t_{1}$ in $\Pi_{\theta}\left(\psi^{\prime}\right)$, it exists a set of tuples $T_{2}$ in $\Pi_{\theta}\left(\psi^{\prime}\right)$, s.t. $t_{1} \notin T_{2}$, that participates with the same values for the attributes in $s\left(\psi^{\prime}\right) \backslash \theta$ (line 12 of Algorithm 6). Thus, if $t[\theta]=t_{1}$, then we have $T_{2}=T[\theta]$. This allows us to conclude that $\left(\delta^{\prime} \bowtie \psi^{\prime}\right)_{\| \theta}=\left(\delta \bowtie \psi^{\prime}\right)_{\| \theta}$. Now, we prove that for $i\left(\delta^{\prime}\right)=i(\delta) \backslash\{t\} \cup T$, the pair $\left(R_{\delta \leftarrow \delta^{\prime}}, F\right)$ is a database, i.e., $R_{\delta \leftarrow \delta^{\prime}}$ satisfies $F^{+}$(note that we did not have to prove this for the last cases, since we always assumed $i\left(\delta^{\prime}\right)$ to be the empty set). For that, we just need to show that $\delta^{\prime}$ satisfies the FDs in $F_{\delta^{\prime}}$, as the other relations in $R_{\delta \leftarrow \delta^{\prime}}$ are the same as those in $R$, which satisfies $F^{+}$. Observe that the only FDs that can be in $F_{\delta^{\prime}}$ are those that have either
(I) only attributes of $s(\delta) \backslash \theta$, or
(II) only attributes of $\theta$, or
(III) attributes of both $s(\delta) \backslash \theta$ and $\theta$.

Regarding (I), since $t[s(\delta) \backslash \theta]=t^{\prime}[s(\delta) \backslash \theta]$ for all $t^{\prime} \in T$, then if no such FD fails for $\delta$, it cannot fail for $\delta^{\prime}$ as well. When it comes to (II), since the values for $\theta$ in any tuple in $i\left(\delta^{\prime}\right)$ are either in a tuple of $\psi$ or are not in $T[\theta]$, as all values in $T[\theta]$ are also in $\psi$, then considering that $\psi$ satisfies $F^{+}$, no FD exclusively with the attributes of $\theta$ can fail for $\delta^{\prime}$ (otherwise it would have failed for $\delta$ ). Finally, for (III), we already assume that no such FD exists in $F^{+}$.

Having proved that (NRT) is satisfied for all the cases, we proved the proposition.

### 5.3 Summary and Discussion

In the previous section, we evaluated analytically our operators of forgetting with respect to the formal properties that were introduced. In this final section, we zoom out on the main content of the dissertation and summarise the key results in order to briefly discuss the most essential points. These include some insights on the operators of forgetting and their properties, as well as their applicability to practical situations and limitations.

Therefore, the purpose of this section is to present an overview of all the work on forgetting carried out in this dissertation, with the ultimate goal of providing a guide
that helps users decide which operators are most appropriate to use in their databases, depending on their needs and requirements. To facilitate the discussion, we present the main results regarding the satisfaction of the properties by the operators in Table 5.1.

Table 5.1: Satisfaction of the properties by the relation forgetting operators. For each operator $f$ and property ( $\mathbf{X}$ ), " $\checkmark$ " means that the operator satisfies the property, while " $\times$ " means that it does not.

|  | (P) | (W) | (I) | (NT) | (SPFD) | (wPFD) | $(\psi-\mathbf{I})$ | (T $\left.{ }^{-}\right)$ | $\left(\mathbf{T}^{+}\right)$ | (NRT) |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| $\mathrm{f}_{n t}$ | $\checkmark$ | $\checkmark$ | $\checkmark$ | $\checkmark$ | $\checkmark$ | $\checkmark$ | $\checkmark$ | $\times$ | $\times$ | $\checkmark$ |
| $\mathrm{f}_{\mathrm{t}}$ | $\checkmark$ | $\times$ | $\checkmark$ | $\times$ | $\times$ | $\checkmark$ | $\checkmark$ | $\checkmark$ | $\checkmark$ | $\times$ |
| $\mathrm{f}_{\mathrm{t}^{-}}$ | $\checkmark$ | $\times$ | $\checkmark$ | $\times$ | $\times$ | $\checkmark$ | $\checkmark$ | $\checkmark$ | $\times$ | $\checkmark$ |
| $\mathrm{f}_{t^{+}}$ | $\checkmark$ | $\times$ | $\checkmark$ | $\times$ | $\times$ | $\checkmark$ | $\checkmark$ | $\times$ | $\checkmark$ | $\checkmark^{*}$ |

${ }^{*}$ Holds under the conditions of Proposition 5.19.

In general, we can divide relation forgetting into two main categories, each aligned with a different view on the operation. The first corresponds to non-transitive forgetting and can be characterised by the property (NT), which imposes equivalence up to the relation to be forgotten between the initial database and the database resulting from the operation. Non-transitive forgetting is especially useful in situations where we want to remove a relation from a database while possibly changing the configuration of the remaining relations, as it guarantees that no extra information is lost and that the new schema is at least as expressive as the initial one under the notion of derivability. On the other hand, this type of forgetting has the disadvantage of leading to the loss of indirect relationships among different attributes in the database that are exclusively guaranteed by the relation to be forgotten. This is due to the fact that (NT) does not permit information from this relation that is not originally in the database to be added to it.

To allow for such kind of forgetting, we have explicitly defined the simple nontransitive forgetting operator $\mathrm{f}_{n t}$.

The second category, which corresponds to transitive forgetting, is primarily characterised by the properties $\left(\mathbf{T}^{-}\right)$and $\left(\mathbf{T}^{+}\right)$together with ( $\mathbf{P}$ ). The idea of transitive forgetting is to add to the database a relation that captures the indirect relationship between the attributes of the relation to be forgotten and those of another relation, before the former is removed, while assuring that the remaining information in the database is kept under equivalence after forgetting. This means that similarly to the non-transitive approach, transitive forgetting accepts transformations at the level of the schema of the database, as long as no additional data is deleted. This, in fact, is guaranteed by the property (P), which turns out to be implied by (NT) (Proposition 5.8).

To some extent, one could think of transitive forgetting as a way to improve nontransitive operators, since it addresses their limitations regarding the preservation of specific indirect relationships that are originally secured by the relation to be forgotten. However, although that is true to a certain degree, both profiles of forgetting have applicability. It is the context and the setting where they are to be used that dictates how
suitable each approach is. In the end, the choice between non-transitive and transitive forgetting is tied to whether or not it is desirable and allowed (e.g., by a court of law) to keep said indirect relationships. This, in turn, depends directly on the information that is to be forgotten.

The operators of transitive forgetting introduced in this document are $f_{t}, f_{t^{-}}$and $f_{t^{+}}$.
Beyond ( $\mathbf{N T}$ ), $\left(\mathbf{T}^{-}\right),\left(\mathbf{T}^{+}\right)$and $(\mathbf{P})$, there are other properties that, even though they are not strong enough to characterise a type of relation forgetting, should not be neglected, as they cover edge cases and therefore help guarantee the desirable behaviour of the operators. For instance, (I) ensures that in case the relation to be forgotten does not belong to the database, all relations remain unchanged under equivalence. Furthermore, $(\psi-\mathbf{I})$ guarantees that if the conditions for transitivity concerning the relation we want to forget with respect to are not satisfied, then the transitive operation is not executed. These properties are satisfied by all the operators, which further justifies their relevance. Even so, we only need to mention them in the context of transitive forgetting, considering that (NT) implies both (I) and ( $\psi-\mathbf{I}$ ) (Propositions 5.8 and 5.12).

Regarding ( $\mathbf{W}$ ), it enforces an upper bound on the result of forgetting, which was shown to be too strong for transitive operators. Therefore, since it is implied by (NT) (Proposition 5.8) and only relevant for non-transitive forgetting, in general we can omit this property.

The properties (SPFD) and (wPFD) impose a minimal set of FDs in the database resulting from forgetting. The first was formalised by taking into account the requirements for non-transitive operators while the latter for the transitive ones.

Finally, (NRT) guarantees that it is impossible to deliberately recover any of the tuples that were in the relation that was forgotten, even if it is known which operator was used to compute forgetting. For this reason, (NRT) plays a pivotal role in forgetting relations. Notably, among the most obvious "absurd" operators that do not satisfy this property, we have those that:

- change the name of the relation to be forgotten;
- add to the relation to be forgotten an arbitrary number of attributes and/or tuples, independently of their values;
- change the values of the relation to be forgotten using a correspondence of one-toone with respect to the initial ones (e.g., deterministic encryption).

Depending on the application, operators that do not obey (NRT) may not be suitable for relation forgetting. In particular, this is the case of $\mathrm{f}_{t}$.

Ultimately, in light of all of the above, in situations where (NRT) is indispensable, in addition to it, we would expect desirable operators to satisfy one of the following sets of properties:

- (NT) and (SPFD);
- (P), either ( $\mathbf{T}^{-}$) or ( $\left.\mathbf{T}^{+}\right),(\mathbf{I}),(\psi-\mathbf{I})$ and (wPFD).

The first set is satisfied by the operator $\mathrm{f}_{n t}$. Conversely, the second one, which includes the properties that characterise transitive forgetting, is satisfied by the operator $\mathrm{f}_{t^{-}}$. In reality, if we restrict the domain of (NRT) to the databases that meet the condition regarding the FDs discussed in Section 4.5 when introducing $f_{t^{+}}$, then this operator also satisfies the second set. Note that this is still a large enough subset of databases in $\mathcal{D}_{\mathcal{A}}$ where the operator can be relevant.

Thus, for these databases, there is a clear trade-off between $f_{t^{-}}$and $f_{t^{+}}$. Whereas the first operator may lead to missing information in the transitive relation, the second may add information to the database that was not previously there. Again, this means that the choice of ideal operator is up to the user, who has to evaluate the requisites of the application at hand.

So, the only operator introduced that does not satisfy one of these sets of properties is $\mathrm{f}_{t}$, as it does not obey (NRT) and also happens to satisfy both ( $\mathbf{T}^{-}$) and ( $\mathbf{T}^{+}$). Nevertheless, this operator is the most adequate to deal with situations where (NRT) is not a requirement for forgetting and we want to preserve some of the indirect relationships that are guaranteed by the relation to be forgotten. This is the case, for example, in circumstances where it is necessary to simplify the database by removing some auxiliary relation.

Still, it is evident that there is room for improvement when it comes to transitive forgetting. For instance, it is important to study:

- operators that satisfy (NRT) and ( $\mathbf{T}^{+}$) for all databases in $\mathcal{D}_{\mathcal{A}}$ (i.e., extend $\mathrm{f}_{t^{+}}$);
- operators that handle FDs more flexibly while still satisfying (wPFD), considering that the approach taken for $f_{t^{-}}$and $f_{t^{+}}$can sometimes be too conservative;
- operators that create transitive relations whose schemas are not only obtained by projecting out the attributes in common for the relation to be forgotten and the one to forget with respect to, provided that these schemas still meet the conditions discussed in Section 4.4.
- methods to compare operators that satisfy either the same or a similar set of properties.

Regarding this last topic, a plausible starting point could be to assess the computational complexity of the operators.

All in all, we can conclude that there is no perfect operator for forgetting relations, and that the choice of the best operator is directly connected to the context where it is to be applied. In summary, this translates into the following. If a transitive relation is not wanted, then $\mathrm{f}_{n t}$ should be the operator of choice. On the contrary, if a transitive relation is desired, then the decision rests on whether (NRT) is required or not. In the
affirmative case, one of the operators $\mathrm{f}_{t^{-}}$or $\mathrm{f}_{t^{+}}$should be used. If not, then $\mathrm{f}_{t}$ would be the most appropriate option.

In any case, once the ideal operator is picked, in order to ease the implementation of the proposed changes in the schema of deployed databases, users can leverage the methods and tools developed in the context of schema evolution (cf. Section 2.6). However, this does not obviate the need for actual implementations of the operators, considering that not only do they propose changes at the level of the schema, but they also define how exactly the data in the database should evolve in the face of those changes. The importance of these implementations is particularly noticeable for the operators $\mathrm{f}_{t^{-}}$and $f_{t^{+}}$, as they manipulate the instances of the relations in a non-trivial way. Therefore, it is essential to develop custom tools that complement those designed for schema evolution in order to automate the application of the operators in databases. In the next chapter, after the conclusions, we will return to this topic in the context of future work.

## Conclusions

In this dissertation, we studied the problem of forgetting relations in relational databases from a theoretical standpoint. This study was motivated by the desire to investigate this operation at a more practical level, such as the one of a database, in order to, e.g., facilitate the implementation of the GDPR's 'right to be forgotten'.

To that end, we started this investigation in Chapter 2, where we overviewed the literature on the operation of forgetting for different knowledge representation formalisms and explored in more depth the practical details of the 'right to be forgotten', examining some of the methods currently used to deal with its demands and commenting on their drawbacks. This review allowed us to identify fundamental requirements an operator of forgetting in databases should abide by, which were later formalised in Chapter 5.

Afterwards, we moved into the domain of relational databases and, in Chapter 3, proposed an alternative formalisation of the relational model, which has the advantage of being better suited to deal with the problem of forgetting than previous formalisations. In addition, we introduced a novel notion of derivability between databases, extending the work of Ausiello et al. [ABM80] and Codd [Cod72] by imposing an extra condition on the ability of both databases to represent information. This notion was later used to define equivalence between databases. The importance of both concepts was witnessed upon the formalisation of the properties of forgetting in Chapter 5, where it was often necessary to compare the content of databases before and after forgetting in a formal way.

In Chapter 4, we looked further into the problem of forgetting relations. We defined the concept of a relation forgetting operator, which was intentionally left as simple and unambiguous as possible to avoid dismissing potential desirable operators early on. Furthermore, having defined the general notion of an operator, we introduced four concrete operators that compute forgetting by means of syntactic manipulations to the original database. Each of these was motivated by a specific set of requirements found in the context of the work carried out in Chapter 2 and illustrated by a running example, which in turn was inspired by a real-world scenario. Moreover, in a first contact with the operators, we showed through some more examples that each has its own unique characteristics and therefore may be more or less adequate than the others for a particular set of applications.

This was later corroborated in our extensive analysis in Chapter 5. For this reason, we split the exposition of the operators into what we identified as the two main views on the operation of forgetting in relational databases, which we called non-transitive and transitive forgetting.

Remarkably, one of the most important similarities between both approaches is that they require that no information in the database other than that exclusively in the relation to be forgotten should be lost with the operation. In fact, this intuition was later formalised by the property ( $\mathbf{P}$ ). Concerning their differences, what really sets these categories apart is the fact that non-transitive forgetting comprises the operators that do not add to the database a relation that preserves the indirect relationships between the attributes in the relation to be forgotten and those of another relation, whereas transitive forgetting does. As such, the later is more in line with forgetting in other knowledge representation formalisms, where indirect relationships are ought to be preserved after forgetting (cf. Example 1.1).

While introducing transitive forgetting, we discussed the exact conditions under which this operation is desirable and the schema that the new relation, dubbed the transitive relation, could take on. We then presented our first operator for this category, which was shown to have a severe drawback in that, in some circumstances, it would allow intentional recovery of some of the tuples in the relation that was forgotten. This observation led us to the introduction of two additional transitive operators, $f_{t^{-}}$and $f_{t^{+}}$, which, in most cases, avoid recovery of forgotten tuples by guaranteeing that for each tuple in the relation to be forgotten there is at least an alternative one that, when replaced with the original, leads to exactly the same database after forgetting. In turn, this implies that even if it is known which operator was used to compute forgetting, it will not be possible to infer any of the forgotten tuples. This idea ended up being formalised in the property (NRT).

Furthermore, we later proved that the operator $f_{t^{-}}$satisfies (NRT) for any database. It does so by identifying the tuples in the relation to be forgotten that cannot be used to compute the transitive relation. The operator $f_{t^{+}}$, on the other hand, only satisfies (NRT) if a specific set of FDs does not belong to the database. In those cases, the operator achieves this result by considering for transitivity tuples that do not actually exist in the relation we want to forget with respect to.

Defining transitive operators that satisfy (NRT) turned out to be more intricate than anticipated, especially in the case of $f_{t^{+}}$, where a similar approach to the one for $f_{t^{-}}$ would not be desirable. In part, this complexity was due to the FDs, which add further constraints to the possible alternatives of each tuple. This forced us to slightly simplify the problem for $\mathrm{f}_{t^{+}}$, which corresponds to one of the limitations of this work.

In chapter 5, we analysed extensively our operators of forgetting. First, we demonstrated that they are indeed operators of forgetting according to the general notion introduced in the previous chapter. Then, we materialised in the form of formal properties the intuitions and requirements that guided the definition of the operators and evaluated
them analytically with respect to those properties, proving that, in general, the operators function in the expected manner. In addition, we demonstrated some relationships between the properties defined. Specifically, we proved that (NT), which corresponds to the property that best characterises non-transitive forgetting, implies important properties for forgetting in general. Perhaps surprisingly, we also showed that despite the apparent simplicity of non-transitive forgetting, the operators fitting in this category do not satisfy (NRT) by default. Overall, this evaluation led to a broad set of theoretical results, which were carefully summarised and discussed in the last part of the chapter. In particular, we stressed the importance of ( $\mathbf{P}$ ) and (NRT) as essential conditions for operators of relation forgetting that deal with legal and data privacy concerns, and highlighted the duality between the operators $f_{t^{-}}$and $f_{t^{+}}$, which makes them suitable for different sets of applications. This reflection prompted the proposal of further enhancements to our operators. At last, we argued that there is no canonical solution to forgetting relations in databases and that the choice of the best operator is tied to the application in mind.

All things considered, we conclude that the theoretical investigation of forgetting relations done in this dissertation, which includes the operators and the properties defined, not only motivates the importance of the operation of forgetting in relational databases but also demonstrates its feasibility in this context. As such, we believe that the work developed in this document lays the foundations for a more comprehensive study of forgetting in relational databases. To this end, in the next section, we suggest directions for future research.

### 6.1 Future Work

Although the first steps towards a complete theory of forgetting in relational databases have been taken, we recognise that much remains to be done. In particular, we draw attention to the following avenues for future developments:

Further refinement of the operators: In the discussion of the operators and their properties in Section 5.3, we suggested improvements to our operators as a means to address some of their limitations. Of those mentioned, we emphasise the importance of investigating ways for $\mathrm{f}_{t^{+}}$to satisfy (NRT) for any database, regardless of its FDs.

In addition, it would also be interesting to study alternative strategies that guarantee (NRT) for transitive operators, other than those employed by $f_{t^{-}}$and $f_{t^{+}}$.

Finally, we reckon it would be worthwhile to develop operators that are more flexible with respect to the schema of the transitive relation and the way FDs are handled.

Computational complexity: The computational complexity of the defined operators was not assessed in this dissertation. However, given its importance to, e.g., compare
operators that satisfy the same or a similar set of properties, it should be considered in future research.

Operator implementation and experimental evaluation: Even though we have shown through the proof of the properties that the operators have the expected semantics, given the theoretical nature of our study and its intended practical applications, it is particularly relevant to implement prototypes for the operators defined and to empirically validate them with case studies inspired by real-world scenarios. We believe that these experiments will bring interesting insights regarding the efficiency and the overhead imposed by the operators, which will propel ways to optimise and further improve them. Ultimately, this line of research will lead to the development of robust tools that automate the process of forgetting in practical systems.

Generalisation of the definition of a relation forgetting operator: Contrary to what usually happens for other knowledge representation formalisms, in general, some of our forgetting operators cannot be iterated, since the order chosen may influence the final result (i.e., for some operators, we cannot guarantee what is commonly referred to as order independence). This is the case, for example, with transitive forgetting, given that the outcome of the operators always depends on two relations. This means that once an iteration is performed, the relation that was forgotten cannot be used in subsequent operations either as $\delta$ or as $\psi$, in the terminology used in the definition of an operator (Def. 4.1), otherwise it will not produce the expected result. To solve this problem, a more general definition for forgetting relations is highly desirable. Initially, a first step could be to extend the current definition to multiple relations $\psi$ and adapt the operators and properties accordingly. Nevertheless, at first glance, this would still not be sufficient to ensure that all the operators commute. Therefore, a more practical solution could be to extend $\delta$ as well and introduce some mechanism that allows establishing mappings for transitivity between the sets $\delta$ and $\psi$. Anyway, future research along these lines must consider the implications that this generalisation of the operators may have on the recovery of forgotten tuples.

Extensions to (NRT): Regarding (NRT), although operators that satisfy this property do not allow the recovery of the exact tuples that were in the relation that was forgotten, under some circumstances, it is still possible to infer some of the values of each tuple beyond those in the transitive relation. In fact, in the worst case, it may be possible to infer all but one value of each original tuple. For this reason, a particularly relevant direction for future work is to formalise properties that extend (NRT) by imposing more constraints on the values of the alternative tuples, as well as to define operators that satisfy those properties.

Furthermore, an interesting follow-up to this research would be to further generalise the operators to take as input a minimum number of alternative tuples to avoid recovery of the original.

Attribute Forgetting: A study on forgetting attributes similar to the one conducted for relations in this dissertation is an important direction for future research. For that, we propose two possible alternatives. The first is to forget an attribute from a single relation, following both the non-transitive and the transitive approaches. On the other hand, the second is to do it for the whole database. In the case of the latter, it seems reasonable to assume that it would be easier to define transitive operators than for the first alternative, considering that without the forgotten attribute in the resulting database, recovery of tuples would no longer be possible. This also suggests that transitive relations resulting from attribute forgetting should not have the attribute in their schema.

After formalising this type of forgetting for a single attribute, it would be worth generalising the operation to support forgetting a set of attributes, in the same vein as what is proposed in this section for forgetting relations.

At last, it would be relevant to compare attribute forgetting operators with relation forgetting operators. Depending on how the first are defined, it may also not be possible to guarantee order equivalence for transitive forgetting, which implies that forgetting a relation with our operators may not be equivalent to iteratively forget every attribute in a relation.

Query adaptation: The study of how queries should evolve in the face of forgetting is a subject that deserves attention in future investigations. In specific, it would be convenient to develop methods that leverage the definition of the operators to establish accurate transformations for the adaptation of queries after forgetting. A possible starting point for this investigation could be to assess the suitability of the tools that propagate schema changes to queries and updates developed in the context of schema evolution (cf. Section 2.6).

Forgetting in relational databases extended with intensional models: To facilitate, for example, interoperability, relational databases are generally equipped with richer conceptual models that further describe the data, their relationships, constraints and semantics (i.e., their intensional meaning). As such, formalising the concept of forgetting for relational databases extended with these models is an area of research that deserves special consideration. In particular, we point towards future work that considers the following intensional models:

- Entity-Relationship (E-R) Model: The study of forgetting in the E-R model should cover forgetting the basic concepts such as entity sets, relationship sets (both binary and non-binary) and attributes, as well as the extended E-R
features, viz. specialisation, generalisation, aggregation, etc. It is possible, however, that the latter may pose extra difficulties due to the added expressivity. In any case, defining proper notions of forgetting in the E-R model could be a challenging task, given that the model lacks precise semantics and formal ways to automatically validate the defined transformations.
- Ontologies expressed in description logics: Since the study of forgetting in DLs is still an active field of research (especially for the most expressive DLs), an interesting starting point for this investigation could be to understand whether the operators proposed in the literature are aligned with the requirements for forgetting discussed in this dissertation. In particular, those of the 'right to be forgotten'. After that, a possible direction could be to examine how forgetting concepts and roles in ontology axioms can be mapped to the database schema so that both the ontology and the schema are consistent.

Overall, the study of forgetting for different configurations of relational databases constitutes an important research topic, which can bring interesting insights and ideas on how the data, schema and ontology should evolve as the result of the operation. Therefore, investigations in this direction are highly encouraged.

## Bibliography

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. Foundations of Databases. AddisonWesley, 1995. isbn: 0-201-53771-0. url: http://webdam.inria.fr/Alice/ (cit. on pp. 76, 77).
[AIR99] J. Albert, Y. E. Ioannidis, and R. Ramakrishnan. "Equivalence of Keyed Relational Schemas by Conjunctive Queries". In: J. Comput. Syst. Sci. 58.3 (1999), pp. 512-534. Doi: 10.1006/jcss.1999.1628. URL: https://doi.org/10.100 6/jcss. 1999.1628 (cit. on pp. 31, 32).
[ALP91] J. Andany, M. Léonard, and C. Palisser. "Management Of Schema Evolution In Databases". In: 17th International Conference on Very Large Data Bases, September 3-6, 1991, Barcelona, Catalonia, Spain, Proceedings. Ed. by G. M. Lohman, A. Sernadas, and R. Camps. Morgan Kaufmann, 1991, pp. 161-170. URL: http: //www.vldb.org/conf/1991/P161.PDF (cit. on p. 22).
[Ara+13] A. Arasu et al. "Orthogonal Security with Cipherbase". In: CIDR 2013, Sixth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 6-9, 2013, Online Proceedings. www.cidrdb.org, 2013. url: http: //cidrdb.org/cidr2013/Papers/CIDR13\\_Paper33.pdf (cit. on p. 21).
[Arm74] W. W. Armstrong. "Dependency Structures of Data Base Relationships". In: Information Processing, Proceedings of the 6th IFIP Congress 1974, Stockholm, Sweden, August 5-10, 1974. Ed. by J. L. Rosenfeld. North-Holland, 1974, pp. 580-583 (cit. on p. 28).
[Atz+82] P. Atzeni et al. "Inclusion and Equivalence between Relational Database Schemata". In: Theor. Comput. Sci. 19 (1982), pp. 267-285. doi: 10.1016/03 04-3975(82)90038-X. URL: https://doi.org/10.1016/0304-3975(82)90038X (cit. on p. 32).
[ABM80] G. Ausiello, C. Batini, and M. Moscarini. "Conceptual Relations between Databases Transformed under Join and Projection". In: Mathematical Foundations of Computer Science 1980 (MFCS'80), Proceedings of the 9th Symposium, Rydzyna, Poland, September 1-5, 1980. Ed. by P. Dembinski. Vol. 88. Lecture

Notes in Computer Science. Springer, 1980, pp. 123-136. doi: 10.1007/BFb0 022499. URL: https://doi.org/10.1007/BFb0022499 (cit. on pp. 31, 34, 91).
[Baa+03] F. Baader et al., eds. The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, 2003. is bn: 0-521-78176-0 (cit. on p. 15).
[BZ05] C. Baral and Y. Zhang. "Knowledge updates: Semantics and complexity issues". In: Artif. Intell. 164.1-2 (2005), pp. 209-243. Doi: 10.1016/j. artint. 2005.01.005. URL: https://doi.org/10.1016/j.artint. 2005.01.005 (cit. on p. 5).
[Boo54] G. Boole. An investigation of the laws of thought on which are founded the mathematical theories of logic and probabilities. Walton and Maberly, 1854 (cit. on p. 9).
[BL04] R. J. Brachman and H. J. Levesque. Knowledge Representation and Reasoning. Elsevier, 2004. isbn: 978-1-55860-932-7. URL: http: //www.elsevier.com/ wps/find/bookdescription.cws\%5C_home/702602/description (cit. on pp. 2, 18).
[Cal+07] D. Calvanese et al. "Tractable Reasoning and Efficient Query Answering in Description Logics: The DL-Lite Family". In: J. Autom. Reason. 39.3 (2007), pp. 385-429. Doi: 10.1007/s10817-007-9078-x. URL: https://doi.org/10 .1007/s10817-007-9078-x (cit. on p. 15).
[Cod70] E. F. Codd. "A Relational Model of Data for Large Shared Data Banks". In: Commun. ACM 13.6 (1970), pp. 377-387. Doi: 10.1145/362384.362685. URL: http://doi.acm.org/10.1145/362384. 362685 (cit. on p. 25).
[Cod72] E. F. Codd. "Further Normalization of the Data Base Relational Model". In: Data base systems 6 (1972), pp. 33-64 (cit. on pp. 31, 34, 91).
[CBN20] U. Comignani, L. Berti-Équille, and N. Novelli. "Discovering Multi-Table Functional Dependencies Without Full Join Computation". In: CoRR abs/2012.06237 (2020). arXiv: 2012.06237. URL: https://arxiv.org/abs/2012.06237 (cit. on pp. 64-66).
[CMZ09] C. Curino, H. J. Moon, and C. Zaniolo. "Automating Database Schema Evolution in Information System Upgrades". In: Proceedings of the 2nd ACM Workshop on Hot Topics in Software Upgrades, HotSWUp 2009, Orlando, FL, USA, October 25, 2009. Ed. by T. Dumitras, I. Neamtiu, and E. Tilevich. ACM, 2009. Doi: 10.1145/1656437.1656444. URL: https://doi.org/10.1145/1656 437.1656444 (cit. on pp. 22, 23).
[Del14] J. P. Delgrande. "Towards a Knowledge Level Analysis of Forgetting". In: Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR 2014, Vienna, Austria, July 20-24, 2014. Ed. by C. Baral, G. D. Giacomo, and T. Eiter. AAAI Press, 2014. url: http: //www.aaai.org/ocs/index.php/KR/KR14/paper/view/7979 (cit. on pp. 5, 11, 14).
[DLS01] P. Doherty, W. Lukaszewicz, and A. Szalas. "Computing Strongest Necessary and Weakest Sufficient Conditions of First-Order Formulas". In: Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001, Seattle, Washington, USA, August 4-10, 2001. Ed. by B. Nebel. Morgan Kaufmann, 2001, pp. 145-154 (cit. on p. 14).
[Ebb85] H. Ebbinghaus. Über das gedächtnis: untersuchungen zur experimentellen psychologie. Duncker \& Humblot, 1885 (cit. on pp. 1, 2).
[EK19] T. Eiter and G. Kern-Isberner. "A Brief Survey on Forgetting from a Knowledge Representation and Reasoning Perspective". In: Künstliche Intell. 33.1 (2019), pp. 9-33. Doi: 10.1007/s13218-018-0564-6. UrL: https://doi.org/ 10.1007/s13218-018-0564-6 (cit. on pp. 2-5, 9-12, 14, 16-19).
[Eyu+18] C. Eyupoglu et al. "An Efficient Big Data Anonymization Algorithm Based on Chaos and Perturbation Techniques". In: Entropy 20.5 (2018), p. 373. doi: 10.3390/e20050373. url: https://doi. org/10.3390/e20050373 (cit. on p. 21).
[GKL16a] R. Gonçalves, M. Knorr, and J. Leite. "The Ultimate Guide to Forgetting in Answer Set Programming". In: Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016. Ed. by C. Baral, J. P. Delgrande, and F. Wolter. AAAI Press, 2016, pp. 135-144. URL: http://www. aaai.org/ocs/ index.php/KR/KR16/paper/view/12849 (cit. on p. 19).
[GKL16b] R. Gonçalves, M. Knorr, and J. Leite. "You Can’t Always Forget What You Want: On the Limits of Forgetting in Answer Set Programming". In: ECAI 2016-22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016). Ed. by G. A. Kaminka et al. Vol. 285. Frontiers in Artificial Intelligence and Applications. IOS Press, 2016, pp. 957-965. doi: 10.3233/978-1-61499-672-9-957. URL: https://doi.org/10.3233/978 -1-61499-672-9-957 (cit. on p. 19).
[Gon+17] R. Gonçalves et al. "When You Must Forget: beyond strong persistence when forgetting in answer set programming". In: CoRR abs/1707.05152 (2017). arXiv: 1707.05152. url: http://arxiv.org/abs/1707.05152 (cit. on p. 19).
[Gon+20] R. Gonçalves et al. "On the limits of forgetting in Answer Set Programming". In: Artif. Intell. 286 (2020), p. 103307. Doi: 10.1016/j.artint.2020.103307. URL: https://doi.org/10.1016/j.artint. 2020.103307 (cit. on pp. 4, 19).
[HTR11] M. Hartung, J. F. Terwilliger, and E. Rahm. "Recent Advances in Schema and Ontology Evolution". In: Schema Matching and Mapping. Ed. by Z. Bellahsene, A. Bonifati, and E. Rahm. Data-Centric Systems and Applications. Springer, 2011, pp. 149-190. doi: $10.1007 / 978-3-642-16518-4 \backslash \_$. URL: https: //doi.org/10.1007/978-3-642-16518-4\\_6 (cit. on pp. 22, 23).
[Hul84] R. Hull. "Relative Information Capacity of Simple Relational Database Schemata". In: Proceedings of the Third ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, April 2-4, 1984, Waterloo, Ontario, Canada. Ed. by D. J. Rosenkrantz and R. Fagin. ACM, 1984, pp. 97-109. dor: 10.1145/588011.58 8027. URL: https://doi.org/10.1145/588011. 588027 (cit. on p. 32).
[KA14] M. Knorr and J. J. Alferes. "Preserving Strong Equivalence while Forgetting". In: Logics in Artificial Intelligence - 14th European Conference, JELIA 2014, Funchal, Madeira, Portugal, September 24-26, 2014. Proceedings. Ed. by E. Fermé and J. Leite. Vol. 8761. Lecture Notes in Computer Science. Springer, 2014, pp. 412-425. doi: 10.1007/978-3-319-11558-0\_29. url: https: //doi.org/10.1007/978-3-319-11558-0\\_29 (cit. on p. 19).
[LLM03] J. Lang, P. Liberatore, and P. Marquis. "Propositional Independence: FormulaVariable Independence and Forgetting". In: J. Artif. Intell. Res. 18 (2003), pp. 391-443. DoI: 10.1613/jair.1113. URL: https://doi.org/10. 1613 /jair. 1113 (cit. on pp. 5, 11).
[Lei17] J. Leite. "A Bird's-Eye View of Forgetting in Answer-Set Programming". In: Logic Programming and Nonmonotonic Reasoning - 14th International Conference, LPNMR 2017, Espoo, Finland, July 3-6, 2017, Proceedings. Ed. by M. Balduccini and T. Janhunen. Vol. 10377. Lecture Notes in Computer Science. Springer, 2017, pp. 10-22. doi: 10.1007/978-3-319-61660-5\_2. url: https://doi.org/10.1007/978-3-319-61660-5\\_2 (cit. on pp. 3, 19).
[LPV01] V. Lifschitz, D. Pearce, and A. Valverde. "Strongly equivalent logic programs". In: ACM Trans. Comput. Log. 2.4 (2001), pp. 526-541. Doi: 10.1145/383779 .383783. url: https://doi org/10.1145/383779. 383783 (cit. on p. 18).
[LR94] F. Lin and R. Reiter. "Forget it!" In: Proceedings of AAAI Fall Symposium on Relevance. 1994, pp. 154-159 (cit. on pp. 3, 5, 11-14).
[Mai83] D. Maier. The Theory of Relational Databases. Computer Science Press, 1983. isbn: 0-914894-42-0. URL: http://web. cecs.pdx. edu/\%5C \% 7Emaier / TheoryBook/TRD.html (cit. on p. 28).
[NS10] A. Narayanan and V. Shmatikov. "Myths and fallacies of "personally identifiable information"". In: Commun. ACM 53.6 (2010), pp. 24-26. doi: 10.11 45/1743546.1743558. URL: http://doi.acm.org/10.1145/1743546.1743558 (cit. on p. 21).
[Pop+11] R. A. Popa et al. "CryptDB: protecting confidentiality with encrypted query processing". In: Proceedings of the 23rd ACM Symposium on Operating Systems Principles 2011, SOSP 2011, Cascais, Portugal, October 23-26, 2011. Ed. by T. Wobber and P. Druschel. ACM, 2011, pp. 85-100. doi: 10.1145/2043556 .2043566. URL: https://doi.org/10.1145/2043556. 2043566 (cit. on p. 21).
[RB06] E. Rahm and P. A. Bernstein. "An online bibliography on schema evolution". In: SIGMOD Rec. 35.4 (2006), pp. 30-31. dor: 10. 1145/1228268. 1228273. URL: https://doi.org/10.1145/1228268.1228273 (cit. on pp. 22, 23).
[Rod92] J. F. Roddick. "Schema Evolution in Database Systems - An Annotated Bibliography". In: SIGMOD Rec. 21.4 (1992), pp. 35-40. Dor: 10.1145/141818.14 1826. URL: https://doi.org/10.1145/141818.141826 (cit. on pp. 22, 23).
[SW18] T. Schaub and S. Woltran. "Answer set programming unleashed!" In: Künstliche Intell. 32.2-3 (2018), pp. 105-108. Doi: 10.1007/s13218-018-0550-z. URL: https://doi.org/10.1007/s13218-018-0550-z (cit. on p. 17).
[SKS11] A. Silberschatz, H. F. Korth, and S. Sudarshan. Database System Concepts, Sixth Edition. McGraw-Hill Book Company, 2011. is bn: 978-0-07-352332-3. URL: https://www.db-book.com/db6/index.html (cit. on pp. 22, 25, 29-31, 51).
[Sjø93] D. Sjøberg. "Quantifying schema evolution". In: Inf. Softw. Technol. 35.1 (1993), pp. 35-44. Doi: 10 . 1016/0950-5849(93) 90027-Z. URL: https : //doi.org/10.1016/0950-5849(93)90027-Z (cit. on p. 22).
[VBO03] G. Van Blarkom, J. J. Borking, and J. E. Olk. "Handbook of privacy and privacy-enhancing technologies". In: Privacy Incorporated Software Agent (PISA) Consortium, The Hague (2003) (cit. on p. 20).
[Wan+08] Z. Wang et al. "Forgetting Concepts in DL-Lite". In: The Semantic Web: Research and Applications, 5th European Semantic Web Conference, ESWC 2008, Tenerife, Canary Islands, Spain, June 1-5, 2008, Proceedings. Ed. by S. Bechhofer et al. Vol. 5021. Lecture Notes in Computer Science. Springer, 2008, pp. 245-257. doi: 10.1007/978-3-540-68234-9\_20. URL: https://doi. org/10.1007/978-3-540-68234-9\%5C_20 (cit. on pp. 5, 15-17).
[Xia+18] G. Xiao et al. "Ontology-Based Data Access: A Survey". In: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden. Ed. by J. Lang. ijcai.org, 2018, pp. 5511-5519. Doi: 10.24963/ijcai.2018/777. URL: https://doi.org/10 .24963/ijcai. 2018/777 (cit. on p. 15).
[ZF06] Y. Zhang and N. Y. Foo. "Solving logic program conflict through strong and weak forgettings". In: Artif. Intell. 170.8-9 (2006), pp. 739-778. doi: 10.101 6/j.artint.2006.02.002. URL: https://doi.org/10.1016/j.artint. 2006.0 2.002 (cit. on p. 5).



[^0]:    This document was created with the (pdf/Xe/Lua)LATEX processor and the NOVAthesis template (v6.9.4).

[^1]:    ${ }^{1}$ In a logic-based formalism, a knowledge base corresponds to a set of formulas, each denoting a statement about a particular world.
    ${ }^{2}$ https://gdpr-info.eu
    ${ }^{3}$ The 'right to be forgotten', corresponding to Article 17, can be found at https://gdpr-info. eu/art-17-gdpr/. Additionally, an explanatory version is available at https://gdpr-info.eu/issues/ right-to-be-forgotten/.

[^2]:    ${ }^{4}$ In fact, type 2 forgetting is more aligned with the area of Belief Change [EK19].
    ${ }^{5}$ A more detailed view on forgetting in answer set programs is given in Section 2.4.

[^3]:    ${ }^{6}$ An interpretation corresponds to the assignment of one value true or false to each variable. Generally, we view interpretations as the set of variables that were assigned true.

[^4]:    ${ }^{7}$ Data pseudonymisation (for example, by replacing personally identifiable information with artificial identifiers) can be a way of complying with the GDPR. According to the Regulation, it should be applied in every system that processes user information, so that no data can be attributed to a specific data subject without the use of additional information, which should be kept separately (see Article $4(5)$ at https://gdpr-info.eu/art-4-gdpr/). This implies that pseudonymisation does not safeguard against later re-identification of the individual the data describes. Conversely, data anonymisation (e.g, by means of generalisation, such that personally identifiable values are replaced with more generic ones), when correctly employed, prevents both direct and indirect identification. By definition, anonymisation is irreversible. Notably, since anonymised data is decoupled from any personal information, it can no

[^5]:    longer be considered personal data and does not fall within the scope of the GDPR (see Recital 26 at https://gdpr-info.eu/recitals/no-26/).

[^6]:    ${ }^{1}$ In fact, some (less expressive) languages do not support role inclusions in their TBoxes.
    ${ }^{2}$ The World Wide Wide Consortium (W3C) is an international community that develops standards for the Web in order to ensure its proper long-term growth (see https://www.w3.org).
    ${ }^{3}$ https://www.w3.org/TR/owl2-overview/

[^7]:    ${ }^{4}$ Informally, answer sets are minimal (classical) models such that all atoms are justified by some rule.

[^8]:    ${ }^{5}$ Strong equivalence in ASP corresponds to logical equivalence in classical logic.
    ${ }^{6}$ Both concrete operators and classes of operators have been proposed in the literature. For simplicity, we adopt the term operator to refer to one and the other.

[^9]:    ${ }^{7}$ See Article 4(5) at https: //gdpr-info.eu/art-4-gdpr/.
    ${ }^{8}$ See Article 32(1)(a) at https://gdpr-info.eu/art-32-gdpr/, Article 25(1) at https://gdpr-info.eu/ art-25-gdpr/ and Recital 28 at https://gdpr-info.eu/recitals/no-28/.
    ${ }^{9}$ See https://gdpr-info.eu/issues/right-to-be-forgotten/.
    ${ }^{10}$ See Recital 26 at https://gdpr-info.eu/recitals/no-26/.

[^10]:    ${ }^{11}$ See Article 17(3) at https://gdpr-info.eu/art-17-gdpr/.

[^11]:    ${ }^{1}$ In the following chapters, whenever it is clear from the context to which term we refer, we use "relation" and "relation instance" interchangeably.

[^12]:    ${ }^{2}$ We follow a usual convention and abbreviate the representation of sets of attributes in FDs to sequences of their elements. For instance, the functional dependency $\{A, B\} \rightarrow\{C\}$ would be written $A, B \rightarrow C$.

[^13]:    ${ }^{3}$ There is extensive work in the literature for derivability and equivalence between database schemas, given the relevance of the topic in the areas of database design, schema evolution, schema integration and schema normalisation, to name a few. For a more comprehensive overview on the subject, the interested reader may refer to [AIR99; Atz+82; Hul84].

[^14]:    ${ }^{1}$ The values in these relations were picked at random for the purpose of this illustrative example.

[^15]:    ${ }^{2}$ This situation is far from unrealistic. For instance, in October 2021, the Portuguese Parliament has approved a bill to punish discrimination in access to credit or insurance against people who have overcome or mitigated serious diseases or disabilities, giving Portuguese citizens the 'right to be forgotten'.

[^16]:    ${ }^{3}$ The natural join operator combines relations by merging pairs of rows, one from each relation, that have equal values on common attributes, into a single row.

[^17]:    ${ }^{4}$ In this dissertation we focus on preserving indirect (transitive) information with respect to a single relation.

[^18]:    ${ }^{5}$ Note that this can happen because we do not impose any restrictions on the schemas of the relations that are part of a database.

[^19]:    ${ }^{6}$ To ease the reading of the algorithm, we abuse notation and say that a tuple $t$ belongs to a relation $r$ $(t \in r)$ or is removed from it $(r \backslash\{t\})$, when $t \in i(r)$ or $i(r) \backslash\{t\}$, respectively. Furthermore, given an ordered set of attributes $\theta=\left(\theta_{1}, \ldots, \theta_{n}\right)$ and a tuple $t=\left(t_{1}, \ldots, t_{n}\right)$, we use $\theta=t$ as a shorthand for $\theta_{1}=t_{1} \wedge \ldots \wedge \theta_{n}=t_{n}$.

[^20]:    ${ }^{7}$ We use the same notation as in Algorithm 5. In addition, given an ordered set of attributes $\theta=\left(\theta_{1}, \ldots, \theta_{n}\right)$ and set of tuples $T=\left\{t^{1}, \ldots ., t^{m}\right\}$, where $t^{i}$ is equal to $\left(t_{1}^{i}, \ldots, t_{n}^{i}\right)$, we use $T=\theta$ to denote $\left(\theta_{1}=t_{1}^{1} \wedge \ldots \wedge \theta_{n}=\right.$ $\left.t_{n}^{1}\right) \vee \ldots \vee\left(\theta_{1}=t_{1}^{m} \wedge \ldots \wedge \theta_{n}=t_{n}^{m}\right)$.

[^21]:    ${ }^{1}$ To simplify, from now on, whenever it is clear from the context, we use " $(\mathbf{X})$ ", where ( $\mathbf{X}$ ) is any property, as a shortcut for "satisfying ( $\mathbf{X}$ )". Therefore, when we write " $(\mathbf{X})$ implies the property $(\mathbf{Y})$ " we mean that every operator that satisfies (X) also satisfies (Y).

[^22]:    ${ }^{2}$ Note that all constraints imposed upon $\delta$ also apply for $\delta^{\prime}$. For instance, $\delta^{\prime}$ must satisfy the same FDs as $\delta$ regarding the original set $F_{D}$. Or, in other words, $D^{\prime}$ must belong to $\mathcal{D}_{\mathcal{A}}$. In addition, by enforcing $s(\delta)=s\left(\delta^{\prime}\right)$, we also guarantee that the domains of the attributes in both relations are the same.

