# On Representing Relationships in Object-Oriented Databases

Dissertation
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
der Universität Dortmund
am Fachbereich Informatik

von
Torsten Polle

Dortmund
2000

| | |
|---|---|
| Tag der mündlichen Prüfung: | 29.10.1999 |
| Dekan: | Prof. Dr. Heinrich Müller |
| Gutachter: | Prof. Dr. Joachim Biskup |
| | Prof. Dr. Norbert Fuhr |

## Abstract

Things in the real world, which surrounds us, do not come as singularity, rather we find them associated. These relationships occur in various forms, for example a person and a car owned by that very person are things associated via the ownership association.

When designing a database for an application, we have to identify and model things pertaining to the application and their relationships. To ease this task, an object-oriented data model offers to model identified things as objects. We model relationships between things as attributes of the corresponding objects. So we introduce for instance for a person and its car objects and define for the “person” object an attribute “owns” holding an reference to the “car” object, or the other way round, i. e., the “car” object receives an attribute holding a reference to the person object. This modelling technique finds its limits when three or more things are associated.

In this work we give a solution to this problem by using first a data model that directly supports relationships, namely the entity-relationship data model, and then by translating results into an object-oriented data model. We propose a transformation called *pivoting* to derive different representations from the initial translation results in a systematic way, and we compare the different representations with respect to their quality. To measure the quality, we give rigorous and precise quality measurements. To do so, we need and subsequently define a formal object-oriented data model and a formal way to tell whether two representations represent the same section of the real world.

## Kurzfassung

Dinge in unserer Umwelt stehen häufig nicht nur für sich allein, sondern sie stehen in Beziehung zu anderen Dingen der Welt. Diese Beziehungen kommen in den unterschiedlichsten Ausprägungen vor, so gibt es beispielsweise Personen und Autos, die diesen Personen gehören.

Beim Entwurf von Informationssystemen müssen wir nun diese Dinge sowie die zugehörigen Beziehungen identifizieren und modellieren. Bei einem objektorientierten Datenmodell lassen sich Beziehungen als Attribute von Objekten modellieren. Auf Personen und Autos bezogen bedeutet das, daß wir Objekte für Personen und Objekte für Autos modellieren und ein Personenobjekt mit einem Attribut versehen, das auf das der Person gehörende Auto verweist. Allerdings scheitert diese Modellierungsart, wenn es sich um Beziehungen mit mehr als drei Objekten handelt.

In dieser Arbeit stellen wir eine Lösung dieses Problems vor, die darauf beruht, ein Datenmodell zu verwenden, das Beziehungen direkt modelliert. Wir benutzen das Entity-Relationship Datenmodell. Eine erfolgreiche Modellierung übersetzen wir anschließend in ein objektorientiertes Datenmodell. Anschließend können wir das Resultat noch weiter bearbeiten und erhalten so unterschiedliche Alternativen zur Darstellung der ursprünglichen Beziehung. Um sicher zu stellen, daß die verschiedenen Alternativen auch den gleichen Ausschnitt aus der realen Welt darstellen, verwenden wir einen formalen Äquivalenzbegriff. Weiterhin untersuchen wir die Alternativen in bezug auf ihre Qualität, wobei wir Qualität mittels formaler Indikatoren messen.

# Acknowledgements

Writing this thesis has been interwoven with my private life. So besides having to thank persons that contributed to it, I want to pay tribute to a number of people that encouraged me, supported me and kept me on the track. I consider as unfair a “ranking”. Therefore I decided to list these persons alphabetically according to their first names in reverse order.

> Ursula Polle (my mother), Thomas Herrschaft (my brother-in-law), Thomas Gehrke, Thilo Baldin, Shuky Sagiv, Rolf Demitz (my first real setter), Ralf Menzel, Petra Baldin, Peter Niebert, Norbert Müller (the judge), Michaela Huhn, Matthias Bunzel (the globe trotter), Martin Schwetje, Lucy (the only female that sometimes obeys my commands), Jürgen Polle (my brother), Joachim Polle (my father), Joachim Biskup (my supervisor), Heikki Mannila, Heike Herrschaft (my sister), Gerrit Bleumer, Elke Rau, Elisabetta Di Nitto (the world’s best driver), Doris Didzoleit, D. K. Knuth (the inventor of TEX and co-author of [KLR88]), David A. Nelson (prefers go-carts to “Rules Royce”), Christiane Meier (♡), Christian Eckert, Birgit Pfitzmann, Barbara Sprick (colleague and flat mate), Annegret Habel, Andreas Pfitzmann, Andrea Schwetje

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

In today's world we are literally inundated by a data flood, which stems from various sources. Because of this flood the data itself becomes meaningless, and therefore we have to harness the flood. We virtually canalise the data into a data stream. In order to gather *information* from this stream, we divert portions, which are sometimes mere data trickles being distilled into information. How do we accomplish this feat? We have to find a model representing the section in the real world under consideration. We obtain such a model by abstracting certain aspects of facts in the reality. The process of obtaining a model starts by analysing the relevant section in the real world.

To put the analysis on formal grounds, we need a formal language to express our models, a *data model*. We call such a data model a *conceptual data model* or *conceptual model* for short. We use the term conceptual, because we want to capture the concepts of the section in the real world. A concrete description of a section expressed in the data model is a *schema*. A schema consists of two components; one for structuring data and one for restricting data complying with the structure description. The actual data stored under a schema is called an *extension*.

Another way in dealing with the data flood is the use of computer systems. Computers are very good at doing things we tell them to do very fast and accurate. So in principle, they offer support for this task. In fact, this was realised a long time ago. So called *database management systems* (DBMS) were invented. What is a database management system? A database management system (cf. [Bis95b]) serves

- to store
- a large amount of data
- persistently,
- dependable,
- shared, and
- efficiently, i. e. supporting queries and updates.

Usually a database management system needs a description of the data to be stored. We need a language to express these descriptions, a *data model*. For this kind of data model we use the term *physical data model*, *physical model* for short, because we give physical specifications such as the types of access to records, indices, ordering and physical placement [BLN86]. Again we call the description a schema, a *physical schema*, and the data to be stored an extension, a *physical extension*. An actual schema and an extension of this schema is called a *database*.

There are a number of data models around, which follow different paradigms. They are grouped according to the paradigm they follow into classes like: hierarchical data models, relational data models, object-oriented data models [Ull88, EN94, AHV95].

Although the mainstream of research has shifted from database design to other fields, object-oriented database design still needs more work [Kim95, ME96]. In this spirit we want to make a contribution to object-oriented database design.

In principle, every physical model can be used in the way a conceptual model is. Indeed, this is done extensively for object-oriented database design. Mostly a variant is used, which drops some of the physical specification features and which is enhanced by more conceptual features. This approach has the advantage that a mapping from the "conceptual" model to the "physical" model is straightforward. A disadvantage of this approach is that in the section in the real world there are entities and relationships among them. But the object-oriented model offers only a natural solution for the representation of binary relationships. However in reality, relationships are not only binary. For that reason, we advocate the use of a conceptual model that offers the concept of a relationship of greater arity than two. Then a conceptual schema expressed in the conceptual model has to be mapped to the physical object-oriented model, when we design for a DBMS following the object-oriented paradigm. As conceptual model we settle on the entity-relationship model (ER-model) initially proposed by Chen [Che76], because it fully meets the aforementioned requirements. The ER-model underwent many revisions and enhancements [ABLV83, CL80, TYF86, KS88, NP88, STH91, GC91, Tha93a, HMPR93, dBL93, JOS93, Hoh93, Gog94, ON95].

This decision provokes a new problem, which we are willing to accept. We have to map a conceptual schema to a physical schema. Therefore Biskup et. al. [BMP96] propose a three step method to design physical object-oriented databases (cf. Fig. 1.1).

- At the first step of conceptual modelling an entity-relationship schema is constructed. This is done by analysing the requirements of an application, which constitutes the section in the real world. The constructed ER-schema can then be augmented to capture a larger section in the real world, or we can integrate existing ER-schemas.

- At the second step of abstract logical formalisation the initial entity-relationship schema is transformed into an abstract object-oriented schema. They use F-logic [KLW95] as basis for the target model. The abstract object-oriented schema can be further improved at this step, in order to exploit object-oriented features.

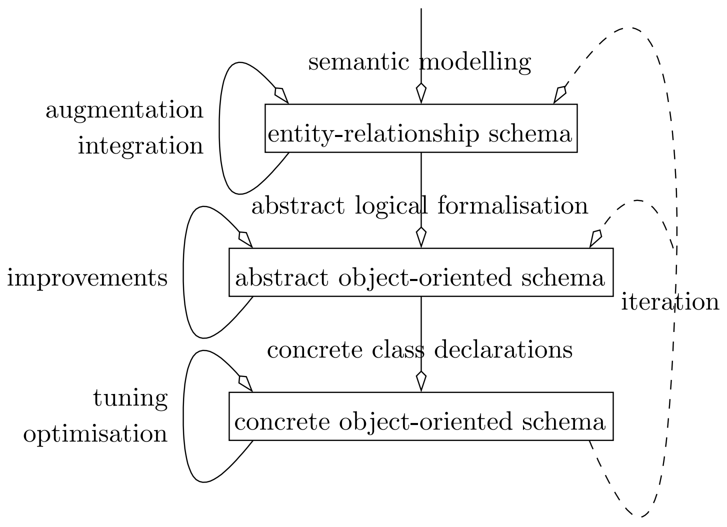

semantic modelling
augmentation
integration
entity-relationship schema
abstract logical formalisation
improvements
abstract object-oriented schema
iteration
concrete class declarations
tuning
optimisation
concrete object-oriented schema

Figure 1.1: A three-step method for object-oriented database design

- At the third step of concrete class declarations the intermediate abstract object-oriented schema is expressed as concrete class declarations of an object-oriented DBMS. We do not deal with this step in this work at all. At this step the schema can be finely tuned and optimised for the specific features offered by the DBMS, which is used.

The reason for the intermediate step is that we can concentrate on particular aspects of design and screen off all disturbing elements. Working in the entity-relationship model at the first step, a designer only has to identify which entities and relationships constitute the application domain without having to think in terms of only binary relationships or to think about the physical placement. She benefits from the graphical representation of ER-schemas as ER-diagrams as well.

Given a real world application, a database designer has to find a description that models the application. In this task she is often confronted with the problem that there are various descriptions for the same application. To facilitate the task of selecting the most appropriate, criteria have been developed to measure the quality of descriptions. The measures are given in different guises; some are mere heuristics, others are formalised. Surely, when we want to open database design to algorithmic treatment, we prefer the latter. Formalising measures presupposes a precise and well-founded data model. F-logic or better a data model based on F-logic appears to be ideally qualified to serve as formal reference model. For at least in principle all important concepts of object-oriented data models concerning data representation and storing, data deduction, integrity maintenance, and high-level querying can be made fully precise within an F-logic based data model. In that way we can apply formal measures for object-oriented data models to schemas obtained in the first step, provided we have a transformation from the ER-model to the object-oriented model. A schema being thus produced can be improved according to the measures.

Semantic constraints play a vital rôle in these improvements, because they are able to capture the dynamics of the real world. They do so by telling which transitions between extensions are permissible.

Often when different applications work in the same section in the real world, the efforts of enforcing semantic constraints are duplicated in every application. Every application has its own code for the enforcement. Fig. 1.2 depicts this situation for two applications.

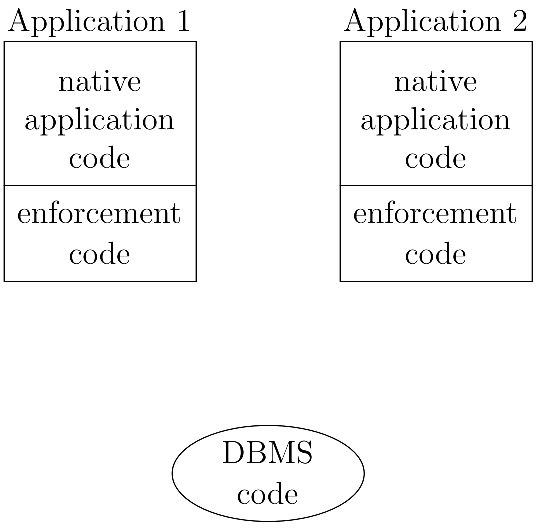


Figure 1.2: Duplicated enforcement code

Since the applications work on the same section in the real world, there must exist semantic constraints that are enforced by both applications. A careful investigation, formalisation and classification might reveal these constraints, because when the enforcement of semantic constraints is embedded in the application code, the same class of semantic constraints occurs in different guises. A situation is desirable where we minimise or even erase the portion of code in an application responsible for the enforcement of semantic constraints. Instead the code is placed in the database management system (Fig. 1.3). A better option still is when the data model can enforce the constraints through the model rather then trough the code.

We deal in this work not with a general classification of semantic constraints in object-oriented data models. Our attention lies on semantic constraints involved in the process of representing relationships in object-oriented data models.

We concentrate in the first step of conceptual modelling more on the structural aspects of the application domain, i. e., we focus on determining the basic entities and their relationships. In the second step of abstract logical formalisation we decide on how to represent the relationships by grouping the corresponding data around objects. This decision and how to reach it on formal grounds is the main concern of this work. Our main means in the examination are onto constraints [Tha93a] and path functional dependencies [Wed89, Wed92], which we combine in this work. Onto constraints can be

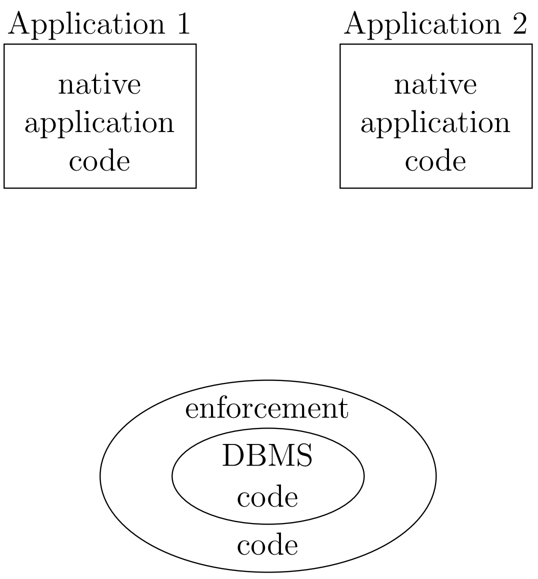


Figure 1.3: Common enforcement code

seen as variants of inclusion dependencies expressed in the relational data model. The notation of an onto constraint has been mentioned by Thalheim [Tha93a] without giving it a name. Path functional dependencies are a generalisation of functional dependencies in the relational data model. To work with these semantic constraints, we give inference rules for the combination of onto constraints and path functional dependencies and show them to be sound and complete.

One criterion to measure the quality of schemas is *redundancy*. This work focuses especially on redundancy in object-oriented representations of relationships. We do not want duplicated data that essentially holds the same information. Our idea of removing redundancy is to introduce a new object storing the redundant data just once and replacing the redundant data everywhere else by a handle to this object. The objects formerly holding the redundant data *share* the data after this process. This process takes place at the data level. What we want is to identify these situations at design time, i. e. at the schema level. A schema is said to have *potential redundancy* if any of its extensions contains redundant data.

A second criterion to assess the quality of a schema is the algorithmic complexity of enforcing semantic constraints. What are the costs of performing updates on existing data and ensuring that the resulting data still is in compliance with the semantic constraints.

We propose a transformation for object-oriented data models, called *pivoting*, which has an impact on both the redundancy potential of a schema and the costs for enforcing semantic constraints under updates. Given our formal data model, we can analyse this impact formally.

This work is organised as follows. In Chapter 2 we lay the foundations for our formal object-oriented data model. In that chapter we present a slight simplified version of

F-logic [KLW95]. We show that the concept of predicate definitions used in predicate logic can be adapted to F-logic. Having set out our basis, we introduce and discuss our formal object-oriented data model in Chap. 3. Based on F-logic, our data model comes with a formal syntax and semantics, which is essential for our investigations. Based on this data model, we formally define three classes of dependencies: onto constraints, class inclusion constraints and path functional dependencies in Chap. 4. Class inclusion constraints had to be invented for technical reasons. Without them we could not present the sound and complete axiomatisation for onto constraints and path functional dependencies. The axiomatisation lays the foundations to make our ensuing results amenable to algorithmic treatment. The inference rules are an extension of Weddell's inference rules [Wed89, Wed92], which deal with path functional dependencies alone in a semantic data model without inheritance. In the annex to that chapter we roughly discuss possible extensions to this work. As we draw our attention to the representation of relationships, we only sketch a canonical transformation from the ER-model to our object-oriented data model and give references for further reading in Chap. 5. When dealing with transformations in data models, the relation between input and output is of concern. We define a framework for the comparison of schemas in Chap. 6. We do this in our logical framework by means of molecule definitions. They enable us to define schema and extension transformations, so called *translation schemes* [MR96, MR98]. Our framework allows us to formulate formulae speaking about schema aspects. Our revision of translation schemes offers to translate these formulae as well. That chapter concludes with a comparison of different approaches to equivalence of databases in object-oriented data models. For the comparison we define the other approaches in our notation. Finally, Chapter 7 brings together the previous results and combines them to define the transformation pivoting and to analyse it. We show when pivoting produces an output equivalent to the input schema. Additionally, we investigate the impact of pivoting on semantic constraints. Of special interest are path functional dependencies. This investigation is coupled with an analysis of how pivoting behaves with respect to the aforementioned quality measures. The results are then compared with the relational model.

Before we embark on our work, we present two examples accompanying us throughout this work. The first one is the "good" example. The second one is the "bad" example. We do not reveal what makes the first one "good" and the second one "bad" until Chap. 7. The examples are both given as entity-relationship diagrams, and we assume the reader to be familiar with the entity-relationship data model. In our notation rectangles (▭) denote entity sets, diamonds (◇) denote relationship sets and ellipses (⬭) denote attributes.

Example 1.1
*We start with the presentation of the "good" example. We give in Fig. 1.4 an example of a conceptual schema an experienced designer would intuitively tend to model with smaller relationships. The ER-diagram reflects the* Assignment *from* Teacher*s and* Assistant*s to* Course*s in combination with the* Date *they take place at, and* Room*s and* Wing*s they are taught in. The semantic constraints are added later in the refinement of the original*

*conceptual schema. They can be already declared in the conceptual schema, but we refrain from doing so, because it would overload the diagram. For every* Course *there is assigned exactly one* Assistant *and it takes place at only one* Date*. At this school each* Teacher *is assigned a fixed* Room*. Additionally, each* Room *is situated in only one* Wing*.*

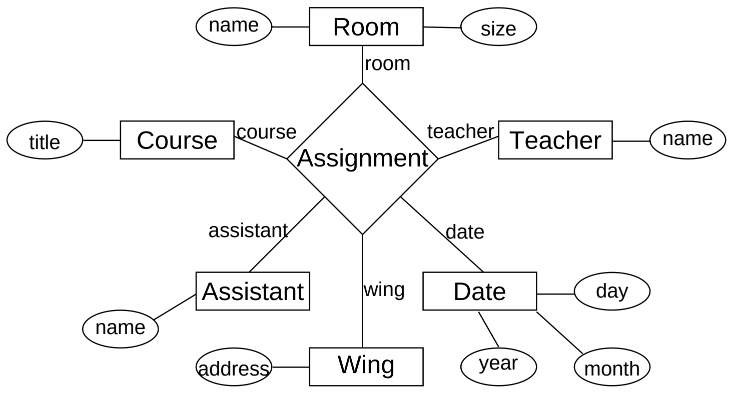


Figure 1.4: An ER-digram of the "good" example

And now to the "bad" example. One might consider the design below as unfit. Still it is chosen with full purpose and is used to demonstrate how it can be improved in Chap. 7.

Example 1.2
*Suppose a database designer, be it an experienced one or not, is given the task to model the following scenario. In a modern office*[1] *a phone belongs to the standard equipment. We assume in this scenario the absence of mobile phones*[2]*. At the end of each month someone has to be charged for the accrued telecommunication costs. For simplicity we assume further that one faculty is charged with the cost of at most one phone identified by its phone number. To make matters more complicated, perhaps we should call at this point for an experienced database designer, the bill is sent to the school for which a person works. Finally, schools are grouped into departments for better handling. For example each department is alloted a stock of phone numbers, preferably some with the same prefix.*

*A first attempt to formalise this scenario is to draw a nice entity-relationship diagram, like the one in Fig. 1.5. The diagram in Fig. 1.5 is far from being complete for various reasons.*

- *It does not capture relationships like "a faculty is charged with the cost of at most one phone".*
- *It does not provide for the recording of phone calls, and*
- *the subsequent billing.*

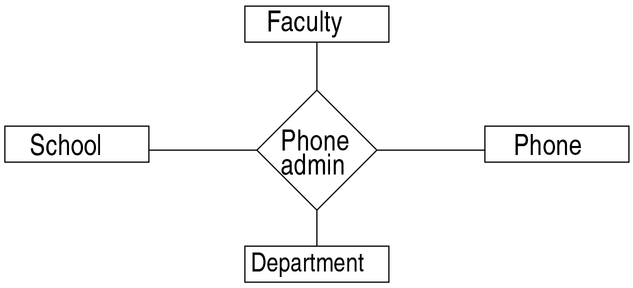


Figure 1.5: An ER-diagram of a phone administration.

*Nevertheless, we are convinced that this design is a realistic one a designer will come forward with, because she starts laying out the four entity sets* Faculty*,* School*,* Phone *and* Department *as formalisation of the corresponding real world entities. In a next step the stated complex (real world) relationship is modelled as relationship set* Phone-admin*. Only in a second attempt this relationship is decomposed into smaller fragments by analysing the relation between the entities. Every faculty member works in exactly one school. She has at most one phone at her disposal. Each school is integrated within exactly one department, and finally exactly one department is responsible for the administration of phone numbers. All these restrictions can be formalised as cardinality constraints [Tha93b], but again we refrain from doing so until Chap. 5.*

[1]As can be found in almost all universities.

[2]Remember it could be a university.

# Chapter 2

# F-Logic

Logic programming has had a profound impact on databases [GM92]. The theory of logic programming has been used to provide a solid theoretical foundation for databases. We follow this approach by using a deductive and object-oriented data model. A number of deductive and object-oriented database languages has been proposed, such as O-logic [Mai86], revised O-logic [KW89], C-logic [CW89], HiLog [CC89], IQL [AK89], IQL2 [Abi90], F-logic[KL89a, KLW95], LOGRES [CCCR$^+$90], LLO [LÖ90], LOL [BM92], Datalog$^{\text{method}}$ [ALUW93], Coral++ [SRSS93], DLT [BB93], Gulog [DT95], Rock & Roll [BFP$^+$95], DM logic [DTM95], Sorted HiLog [CK95] and ROL [Liu96]. From this palette we choose F-logic as the basis for our data model, because it is a very powerful language, although it does not treat sets as first class citizens [Liu96]. This choice is motivated by the appropriate combination of expressiveness and flexibility, which F-logic offers. In particular we can base our investigations on the precise and formal semantics. For at least in principle, all important aspects concerning data representation and storing, data deduction, integrity maintenance and high-level querying can be made fully precise within F-logic. So it is an ideal tool for our needs.

F-logic is a rich and powerful language that uniformly combines the highlights of deductive and object-oriented databases. Incorporating the object-oriented paradigm, F-logic accounts for most of the structural aspects of object-oriented languages.

- F-logic supports object identity by means of logical object identifiers.
- Objects are built from frames, so they have an internal structure and thus can form complex objects.
- Objects with a common structure can be grouped into classes. The classes can build a class hierarchy.
- Signatures as types of operations are inherited to subclasses.

On the side of the deductive paradigm F-logic has

- a syntax, which, among other things, allows the user to explore schema and data using the same declarative language,

- a model theoretic semantics, which offers a precise semantics and is supplemented by a sound and complete proof theory, and
- a sound and complete operational semantics.

For our purposes we slim down F-logic a little bit. We will only deal with non-inheritable data expressions.

## 2.1 Syntax

F-logic comes with a higher-order syntax, but the underlying semantics remains first-order. An F-logic language is defined over an alphabet. It consists of object constructors, which play the rôle of functional symbols, predicate symbols, variables, some auxiliary symbols and the usual logical connectives and quantifiers.

Definition 2.1 (Alphabet)
*The* alphabet $\mathcal{A}$ *of an F-logic language,* $\mathcal{L}$*, consists of,*

- *a set of* object constructors $\mathcal{F}$*,*
- *a set of* predicate symbols $\mathcal{P}$ *including the equality predicate* $\overset{\circ}{=}$*,*
- *an infinite set of* variables $\mathcal{V}$*,*
- *auxiliary symbols, such as,* (, ), [, ], $\rightarrow$, $\twoheadrightarrow$, $\Rightarrow$, $\Rrightarrow$*, etc., and*
- *usual logical connectives and quantifiers,* $\vee$, $\wedge$, $\neg$, $\longleftarrow$ , $\forall$, $\exists$, $\exists^{=1}$.

*The set of object constructors* $\mathcal{F}$ *and the set of predicate symbols* $\mathcal{P}$ *are disjoint,* $\mathcal{F} \cap \mathcal{P} = \emptyset$*. We denote the union* $\mathcal{F} \cup \mathcal{P}$ *of the set of object constructors* $\mathcal{F}$ *and of predicate symbols* $\mathcal{P}$ *by* $\mathcal{S}$*.*

Each object constructor has an *arity* — a non-negative integer that determines how many arguments this constructor can take. Constructors of arity 0 play the rôle of constant symbols; constructors of arity greater than 1 are used to construct larger terms out of simpler ones. Such *id-terms* are first-order terms composed of object constructors and variables, as in predicate calculus.

Definition 2.2 (id-Term)
*Let* $\mathcal{A}$ *be an alphabet.* Id-terms *over the alphabet* $\mathcal{A}$ *are built out of object constructors and variables.*

- *An object constructor* $c \in \mathcal{F}$ *of arity* 0 *is an* id-term.
- *A variable* $V \in \mathcal{V}$ *is an* id-term.
- *If* $f \in \mathcal{F}$ *is an object constructor of arity* $n$*, and* $T_1, \ldots, T_n$ *are id-terms, then* $f(T_1, \ldots, T_n)$ *is an* id-term.

- *An id-term is called* ground *if it is variable-free. The set of all ground id-terms is denoted by $U(\mathcal{F})$. This set is commonly known as* Herbrand Universe.

Ground id-terms play the rôle of logical object identifiers. They are an abstraction of the implementational concept of physical object identity.

We have the convention for alphabets and id-terms that symbols set in sans-serif font are ground and symbols set in *italic font* beginning with lower-case letters are ground while symbols beginning with capital letters denote id-terms that may be non-ground.

Example 2.3
*The terms $X$,* bob*,* alice*,* child(bob)*,* child($X$) *are id-terms. Following the convention, here* bob*,* alice*,* child(bob) *are ground id-terms and $X$,* child($X$) *are non-ground id-terms, because both contain variable $X$.*

In predicate calculus, the atom formulae are facts. In F-logic the atom formulae are built from *molecular formulae*, which state facts about objects and classes. These molecular formulae are of different forms. Some take the form "id-term operator id-term", so-called *is-a assertions*. Some take the form of frames, where methods take the rôle of slots. These are called *object molecules* and define the method values for objects and their signatures. For our purposes we modify F-logic slightly. In particular, we shall not use so-called inheritable data expressions, only non-inheritable data expressions. Therefore we do not introduce inheritable data expressions at all. The interested reader is referred to [KLW95] for more detail on the distinction between inheritable and non-inheritable data expressions. As both form an orthogonal feature of F-logic, the modified definitions do not raise any problems.

Definition 2.4 (Molecular Formula)
*Let $\mathcal{A}$ be an alphabet. A* molecule *over alphabet $\mathcal{A}$ in F-logic is one of the following statements:*

1. *An* is-a assertion *of the form $C :: D$ or of the form $O : C$, where $C$, $D$ and $O$ are id-terms over alphabet $\mathcal{A}$.*

2. *An* object molecule *of the form $O$[";"-separated list of* method expressions]*, where $O$ is an id-term over the alphabet $\mathcal{A}$. A* method expression *can be either a* data expression *or a* signature expression.

   - Data expressions *take one of the following two forms:*
     - *a* scalar *data expression ($k \geq 0$):*

$$M \;@\, Q_1, \ldots, Q_k \rightarrow T \;\;,$$

     - *a* set-valued *data expression ($l, m \geq 0$):*

$$M' \;@\, R_1, \ldots, R_l \twoheadrightarrow \{S_1, \ldots, S_m\} \;\;,$$

   *where $M$, $Q_1, \ldots, Q_k$, $T$, $M'$, $R_1, \ldots, R_l$, and $S_1, \ldots, S_m$ are id-terms over alphabet $\mathcal{A}$.*

- Signature expressions *also take two forms:*
  - *a* scalar *signature expression* $(n, r \geq 0)$*:*

$$M \; @ \, V_1, \ldots, V_n \Rightarrow (A_1, \ldots, A_r) \ ,$$

  - *a* set-valued *signature expression* $(s, t \geq 0)$*:*

$$M' \; @ \, W_1, \ldots, W_s \Rrightarrow (B_1, \ldots, B_t) \ ,$$

*where $M$, $V_1, \ldots, V_n$, $A_1, \ldots, A_r$, $M'$, $W_1, \ldots, W_s$, and $B_1, \ldots, B_t$ are id-terms over alphabet $\mathcal{A}$.*

3. *A* predicate molecule *(abbrev. P-molecule) of the form $p(T_1, \ldots, T_n)$, where $p \in \mathcal{P}$ is an n-ary predicate symbol and $T_1, \ldots, T_n$ are id-terms over alphabet $\mathcal{A}$.*

Is-a assertions of the form $C :: D$ in (1) state that class $C$ is a subclass of class $D$ and of the form $O : C$ that object $O$ is in the extension of class $C$.

Example 2.5
*The following is-a assertion states that the object representing person Bob is in class* Faculty*, thereby modelling the fact that Bob is employed as faculty. The next one states that the object representing person Alice is in class* Person*. The last is-a assertion indicates that every faculty is also a person.*

bob : Faculty
alice : Person
Faculty :: Person

In object molecules in (2) the id-term $O$ denotes an object. In data expressions, methods are either invoked as scalar methods or set-valued methods on object $O$. Single headed arrows, $\rightarrow$ and $\Rightarrow$, indicate that method $M$ denotes a scalar function. The double headed arrows, $\twoheadrightarrow$ and $\Rrightarrow$, indicate that the corresponding method is set-valued.

The return values of the method invocation on object $O$ are represented by $T$ and $S_1, \ldots, S_m$ for scalar and set-valued methods, respectively. In signature expressions, the id-terms $V_1, \ldots, V_n$ and $W_1, \ldots, W_s$ represent the types of the arguments a method is invoked with. The types of the return values of the method invocation are represented by $A_1, \ldots, A_r$ and $B_1, \ldots, B_t$. The results returned by the respective functions must belong to all result types.

Example 2.6
*The object molecule* bob[children @ alice $\twoheadrightarrow$ {john}] *contains the set-valued data expression* children @ alice $\twoheadrightarrow$ {john}*. The object molecule states that the invocation of the respective function for method* children *on the object corresponding to id-term* bob *with an argument object corresponding to* alice *returns at least the object corresponding to id-term* john*. The signature for method* children *on class* Person *might be* Person[children @ Person $\Rrightarrow$ (Person)]*. It demands that the invocation of the respective function for method* children *on objects of the class corresponding to id-term* Person

*takes objects of the class corresponding to id-term* Person *and returns objects of the class corresponding to id-term* Person*.*

*The polymorphism of F-logic allows the simultaneous declaration of object molecule* $\mathsf{Person}[\mathsf{children}\ @\!\!\Rightarrow\!\!\!\!\Rightarrow (\mathsf{Person})]$. *Here method* children *is invoked without arguments. This declaration might be appropriate if method* children *without arguments is defined to return the union of the results when the method is invoked with arguments, thereby defining the value.*

*In most cases it is possible to relax the strict syntax. We shall not define this relaxations formally rather we give an example for it, to give the reader a flavour.*

$$\begin{aligned}&\mathsf{bob} : \mathsf{Faculty}, \mathsf{alice} : \mathsf{Person}, \mathsf{john} : \mathsf{Person},\\&\mathsf{Faculty} :: \mathsf{Person},\\&\mathsf{bob}[\mathsf{children}\ @\ \mathsf{alice} \twoheadrightarrow \{\mathsf{john}\}],\\&\mathsf{bob}[\mathsf{children}\ @\!\twoheadrightarrow \{\mathsf{john}\}]\end{aligned}$$

*is on a par with*

$$\begin{aligned}&\mathsf{Faculty} :: \mathsf{Person},\\&\mathsf{bob} : \mathsf{Faculty}[\mathsf{children}\ @\ \mathsf{alice} : \mathsf{Person} \twoheadrightarrow \mathsf{john} : \mathsf{Person};\\&\qquad\qquad \mathsf{children} \twoheadrightarrow \mathsf{john}]\ .\end{aligned}$$

Sometimes we normalise molecules by breaking them apart into their *constituent atoms.* The semantics of F-logic ensures that an object molecule is logically on a par with its set of constituent atoms.

Definition 2.7 (Constituent Atom)
*Let $G$ be a molecule.*

- *If the molecule $G$ is an is-a assertion or P-molecule, it is a* constituent atom*.*
- *If the molecule $G$ is an object molecule of the form $G = O[method\ expressions]$, the* constituent atoms *are:*
  - *For every scalar data expression $M\ @\ Q_1, \ldots, Q_k \rightarrow T$ in molecule $G$, the corresponding* constituent atom *is:*

    $$O[M\ @\ Q_1, \ldots, Q_k \rightarrow T]\ .$$

  - *For every set-valued data expression $M'\ @\ R_1, \ldots, R_l \twoheadrightarrow \{S_1, \ldots, S_m\}$ in molecule $G$, the corresponding* constituent atoms *are:*

    $$\begin{aligned}&O[M'\ @\ R_1, \ldots, R_l \twoheadrightarrow \{\,\}]\ ,\ \ \textit{and}\\&O[M'\ @\ R_1, \ldots, R_l \twoheadrightarrow S_i]\ \textit{for}\ i \in \{1, \ldots, m\}.\end{aligned}$$

– *For every signature expression*[1] $M \ @ \ V_1, \ldots, V_n \Rrightarrow (B_1, \ldots, B_t)$ *in molecule* $G$*, the corresponding* constituent atoms *are:*

$$O[M \ @ \ V_1, \ldots, V_n \Rrightarrow (\,)] \ , \ \textit{and}$$
$$O[M \ @ \ V_1, \ldots, V_n \Rrightarrow B_i] \ \textit{for} \ i \in \{1, \ldots, t\}.$$

The syntactic restrictions on id-terms when they occur as results of set-valued methods guarantee that they stand for only elements not sets of elements. The semantics of these id-terms can be seen as being iterative and thus ensuring the first-order semantics.

In F-logic, it follows from the syntax that every logical id-term can denote either an entity or a method, depending on the syntactic position of this id-term within the formula. An entity in this context should be taken to mean it can either be a class or an object. In an occurrence as a method, this id-term denotes either a scalar function or a set-valued function. The type of the invocation can be determined by the context. This behaviour might be appropriate for some applications. We do not need this freedom and, consequently, impose a restriction on id-terms in Chap. 3.

With molecular formulae as ingredients, we can build larger formulae, so-called F-formulae.

Definition 2.8 (F-Formula)
*Let $\mathcal{A}$ be an alphabet.* F-formulae *over the alphabet $\mathcal{A}$ are built up from simpler F-formulae by means of logical connectives and quantifiers:*

- *Molecular formulae over the alphabet $\mathcal{A}$ are* F-formulae *over the alphabet $\mathcal{A}$,*
- $\alpha \vee \beta$*,* $\alpha \wedge \beta$*,* $\neg\alpha$ *are* F-formulae *over the alphabet $\mathcal{A}$, if so are $\alpha$ and $\beta$, and*
- $(\forall X)\,\alpha$*,* $(\exists Y)\,\beta$*,* $(\exists^{=1} Z)\,\gamma$ *are* F-formulae *over the alphabet $\mathcal{A}$, if so are $\alpha$, $\beta$, $\gamma$ and $X, Y, Z \in \mathcal{V}$ are variables.*

If $\mathcal{S}$ is the set of symbols of the alphabet $\mathcal{A}$ and $\alpha$ is an F-formula over the alphabet $\mathcal{A}$, then we say $\alpha$ is an $\mathcal{S}$-formula. This parlance is intensively used in Sec. 2.5.

Throughout this work, we shall often use the implication connective, "$\longleftarrow$". In F-logic, this connective is defined as usual. The formula $\alpha \longleftarrow \beta$ stands for $\alpha \vee \neg\beta$.

Definition 2.9 (F-Logic Language)
*Let $\mathcal{A}$ be an alphabet. The* F-logic language *$\mathcal{L}$ over the alphabet $\mathcal{A}$ is the set of all F-formulae over the alphabet $\mathcal{A}$.*

Example 2.10
*The construct*

$$P[\mathsf{children} \twoheadrightarrow C] \ \longleftarrow \ P[\mathsf{children} \ @ \ S \twoheadrightarrow C]$$

*is an F-formula with $P$, $C$ and $S$ as variables and* children *as object constructor according to our convention.*

[1]The arrow $\Rrightarrow$ in the signature expression $M \ @ \ V_1, \ldots, V_n \Rrightarrow (B_1, \ldots, B_t)$ stands for both kinds of arrows either $\Rightarrow$ or $\Rightarrow\!\!\!\Rightarrow$. So with the signature expression above the statement is valid for both scalar and set-valued signature expressions.

In the sequel we deal with certain concepts, so we define abbreviations for them here.

Definition 2.11 (Literal, F-Horn-rule)
*Let $\mathcal{L}$ be an F-logic language.*

- *A* literal *is either a molecular formula, called* positive literal*, or a negation of a molecular formula, called* negative literal.
- *A formula head $\longleftarrow$ body where head is a positive literal and body is a conjunction of positive literals is called an* F-Horn-rule*, corresponding to their counterpart, Horn-rules, in classical logic.*

The formula in Exam. 2.10 is an F-Horn-rule.

Definition 2.12 (Free and Bound Occurrence of Variables)
*Let $\alpha$, $\beta$ be F-formulae.*

- *If $\alpha$ is a molecule, all occurrences of variables in $\alpha$ are free.*
- *Let $X$ be a variable occurring free in $\alpha$ or $\beta$.*
  - *Variable $X$ occurs free in $\alpha \vee \beta$, $\alpha \wedge \beta$, $\neg\alpha$, $(\forall Y)\alpha$, $(\exists Y)\alpha$, $(\exists^{=1} Y)\alpha$, where $X \neq Y$.*
  - *Variable $X$ occurs bound in $(\forall X)\alpha$, $(\exists X)\alpha$, $(\exists^{=1} X)\alpha$.*

Definition 2.13 (Closed and Classical F-Formula)

- *An F-formula is a* closed F-formula *or* sentence*, if all variable occurrences in it are bound.*
- *An F-formula is a* classical F-formula*, if all molecules occurring in it are P-molecules.*

## 2.2 Semantics

As mentioned before, F-logic is furnished with a first-order, model-theoretic semantics. Semantic structures that describe the potential worlds are called *F-structures*. These structures give meaning to classes, objects, methods and alike. In preparation for the definition of F-structures, Def. 2.15, we need the following auxiliary definitions.

Definition 2.14
*Let $U$, $V$ be a pair of sets, and $\prec_U$ and $\prec_V$ be partial orders defined on $U$ and $V$, respectively.*

- *The set of all* total *functions $U \mapsto V$ is denoted by* $\mathrm{Total}(U, V)$.
- *Similarly, the set of all* partial *functions $U \mapsto V$ is denoted by* $\mathrm{Partial}(U, V)$.

- *The expression* PartialAntiMonotone$_{\prec_U,\prec_V}(U,V)$ *stands for the set of partial* anti-monotonic[2] *functions.*
- *The* power-set *of a set $U$ is denoted by $\wp(U)$.*
- $\wp_{\uparrow}(U)$ *is the* set of all upward-closed[3] subsets of $U$.
- *Given the collection of sets $\{S_i\}_{i\in\mathbb{N}}$ parameterised by the natural numbers, $\mathbb{N}$, $\prod_{i=0}^{\infty} S_i$ stands for the* Cartesian product *of the $S_i$'s, i. e., the set of all infinite tuples, $\langle s_0,\ldots,s_n,\ldots\rangle$.*

Now we are ready to define F-structures.

Definition 2.15 (F-Structure)
*Let $\mathcal{L}$ be an F-logic language. An* F-structure *is a tuple*

$$\mathcal{I} = \langle U, \prec_U, \in_U, \mathcal{I}_{\mathcal{F}}, \mathcal{I}_{\mathcal{P}}, \mathcal{I}_{\rightarrow}, \mathcal{I}_{\twoheadrightarrow}, \mathcal{I}_{\Rightarrow}, \mathcal{I}_{\Rrightarrow}\rangle \quad .$$

*Here*

- *$U$ is the domain of $\mathcal{I}$ (sometimes denoted $\mathcal{I}(U)$),*
- *$\prec_U$ is a partial order on $U$ for the class hierarchy, we write $a \preceq_U b$ if $a \prec_U b$ or $a = b$,*
- *$\in_U$ is a binary relation over $U$ for the population of classes, such that if $u \in_U v$ and $v \prec_U w$, then $u \in_U w$.*
- *$\mathcal{I}_{\mathcal{F}}$ is a mapping, $\mathcal{F} \mapsto \bigcup_{i=0}^{\infty} \mathrm{Total}(U^i, U)$, for the interpretation of object constructors (sometimes denoted $\mathcal{I}(\mathcal{F})$),*
- *$\mathcal{I}_{\mathcal{P}}(\cdot)$ is a relation on $U$, $\mathcal{I}_{\mathcal{P}}(p) \subset U^n$, for any $n$-ary predicate symbol $p \in \mathcal{P}\backslash\{\overset{\circ}{=}\}$, and $\mathcal{I}_{\mathcal{P}}(\overset{\circ}{=}) := \{\langle a,a\rangle \mid a \in U\}$ (sometimes denoted $\mathcal{I}(\mathcal{P})$),*
- *$\mathcal{I}_{\rightarrow}$ is a mapping, $\mathcal{I}_{\rightarrow}\colon U \mapsto \prod_{k=0}^{\infty} \mathrm{Partial}(U^{k+1}, U)$, for the interpretation of scalar methods,*
- *$\mathcal{I}_{\twoheadrightarrow}$ is a mapping, $\mathcal{I}_{\twoheadrightarrow}\colon U \mapsto \prod_{k=0}^{\infty} \mathrm{Partial}(U^{k+1}, \wp(U))$, for the interpretation of set-valued methods,*
- *$\mathcal{I}_{\Rightarrow}$, $\mathcal{I}_{\Rrightarrow}$ are mappings, $U \mapsto \prod_{k=0}^{\infty} \mathrm{PartialAntiMonotone}_{\vec{\preceq}_{U^{k+1}},\subset}(U^{k+1}, \wp_{\uparrow}(U))$, for the interpretation of signatures, where $\vec{\preceq}_{U^{k+1}}$ is the extension of $\prec_U$ on $U^{k+1}$ with: for any $(u_1,\ldots,u_{k+1}), (v_1,\ldots,v_{k+1}) \in U^{k+1}$, we write*

  $$(u_1,\ldots,u_{k+1}) \vec{\preceq}_{U^{k+1}} (v_1,\ldots,v_{k+1})$$

  *if $u_i \prec_U v_i$ for all $i \in \{1,\ldots,k+1\}$.*

[2]For the partial function $\rho : U \mapsto V$, anti-monotonic means that if $u, v \in U$, $u \prec_U v$, and $\rho(v)$ is defined, then $\rho(u)$ is also defined and $\rho(u) \succ_V \rho(v)$.

[3]A subset $U'$ of $U$ is upward-closed, if for all $v \in U'$, $u \in U$ with $v \prec_U u$, then $u \in U'$.

The domain $U$ constitutes the universe of discourse. Its elements stand for classes, objects and methods. The elements of $U(\mathcal{F})$, ground id-terms, play the rôle of logical object identifiers. They are interpreted by the elements in $U$ via the mapping $\mathcal{I}_\mathcal{F}\colon \mathcal{F} \mapsto \bigcup_{i=0}^{\infty} \text{Total}(U^i, U)$. This mapping interprets each $k$-ary object constructor by a total function $U^k \mapsto U$. For $k = 0$, $\mathcal{I}_\mathcal{F}(f)$ can be identified with an element of the domain $U$.

The semantic counterpart of the subclass hierarchy is put up by the ordering $\prec_U$ on the domain $U$. An expression $a \prec_U b$ is interpreted as a statement that element $a$ is a subclass of element $b$. To model class memberships, we use the binary relation $\in_U$ over the domain $U$. An expression $a \in_U b$ is interpreted as statement that element $a$ is in the extension of element $b$.

Id-terms can also denote methods. These methods are interpreted by the mappings $\mathcal{I}_{\rightarrow}$, $\mathcal{I}_{\twoheadrightarrow}$, $\mathcal{I}_{\Rightarrow}$, $\mathcal{I}_{\Rrightarrow}$. As methods are invoked on host objects with a list of arguments, either as scalar methods or as set-valued methods depending on the context, an F-structure has to attach appropriate functions to each method in order to assign meaning to methods.

To allow the use of variables in id-terms at method positions, id-terms by themselves are not interpreted as functions, instead the elements of $U$ associated with these id-terms via mapping $\mathcal{I}_\mathcal{F}$ are interpreted as tuples of functions. This tuple provides exactly one function for each arity. This is because of the polymorphism of F-logic, since each method can have different arities.

Furthermore, F-logic does not demand that every method invocation returns a proper value. This is formally captured by the use of partial functions.

In addition to different arities, every method can be invoked as a scalar or set-valued function. This is achieved by interpreting the set-valued incarnations of methods separately. The interpretation of set-valued methods returns sets instead of just elements.

The arity of the interpreting function is one greater than the arity of the corresponding method because the former takes the host object as argument.

The mappings $\mathcal{I}_{\Rightarrow}$ and $\mathcal{I}_{\Rrightarrow}$ specify the type of a method. This type is a functional type, since methods are interpreted as functions. Therefore the allowed arguments and results of method invocations are specified. In addition, the specification must allow for the polymorphism. For the same reason we mentioned above, mappings relate objects of the domain $U$ to each other. The anti-monotonicity and upward-closeness are needed to ensure that when the result type is specified, all its superclasses are a valid result type as well.

The definition of an F-structure $\mathcal{I}$ depends on the set of symbols $\mathcal{S}$ underlying the F-logic language $\mathcal{L}$. Therefore we say sometimes an F-structure $\mathcal{I}$ is an $\mathcal{S}$-structure.

An F-structure is an extension of a structure in classical logic. We can restrict an F-structure to the classical components. Additionally, we can restrict an F-structure to interpret only symbols in a subset of the original set of symbols.

Definition 2.16 (Restriction and Classical Restriction)
*Let $\mathcal{I} = \langle U, \prec_U, \in_U, \mathcal{I}_\mathcal{F}, \mathcal{I}_\mathcal{P}, \mathcal{I}_{\rightarrow}, \mathcal{I}_{\twoheadrightarrow}, \mathcal{I}_{\Rightarrow}, \mathcal{I}_{\Rrightarrow}\rangle$ be an $\mathcal{S}'$-structure, and $\mathcal{S} \subset \mathcal{S}'$ be a set of symbols.*

- *The* restriction *$\mathcal{I}\restriction_\mathcal{S}$ is defined as $\mathcal{I}$ except for the interpretation of $\mathcal{I}\restriction_\mathcal{S}(\mathcal{F})$ and*

$\mathcal{I}\upharpoonright_{\mathcal{S}}(\mathcal{P})$.

- $\mathcal{I}\upharpoonright_{\mathcal{S}}(\mathcal{F})(f) := \mathcal{I}_{\mathcal{F}}(f)$ *for all object constructors* $f \in \mathcal{S}$.
- $\mathcal{I}\upharpoonright_{\mathcal{S}}(\mathcal{P})(p) := \mathcal{I}_{\mathcal{P}}(p)$ *for all predicate symbols* $p \in \mathcal{S}$.

- *The* classical restriction $\mathcal{I}\lceil_{\mathcal{S}} := \langle \mathcal{I}\lceil_{\mathcal{S}}(U), \mathcal{I}\lceil_{\mathcal{S}}(\mathcal{F}), \mathcal{I}\lceil_{\mathcal{S}}(\mathcal{P})\rangle$ *is defined as follows:*
  - $\mathcal{I}\lceil_{\mathcal{S}}(U) := U$.
  - $\mathcal{I}\lceil_{\mathcal{S}}(\mathcal{F})(f) := \mathcal{I}_{\mathcal{F}}(f)$ *for all object constructors* $f \in \mathcal{S}$.
  - $\mathcal{I}\lceil_{\mathcal{S}}(\mathcal{P})(p) := \mathcal{I}_{\mathcal{P}}(p)$ *for all predicate symbols* $p \in \mathcal{S}$.

In the definition of an F-structure the relationship between $\mathcal{I}_{\rightarrow}$ and $\mathcal{I}_{\Rightarrow}$, or $\mathcal{I}_{\twoheadrightarrow}$ and $\mathcal{I}_{\Rightarrow\!\!\!\!\!\Rightarrow}$, respectively, is not fixed. Although $\mathcal{I}_{\Rightarrow}$ and $\mathcal{I}_{\Rightarrow\!\!\!\!\!\Rightarrow}$ specify the argument and result types, so far this is not enforced. In Sect. 2.4 we will capture this on a meta level. Later on we will use semantic constraints as defined in Sect. 3.1.

Now we come to the satisfaction of F-formulae by F-structures. Therefore we have to assign variables to elements of the domain $U$ and, subsequently, extend this assignment over id-terms.

Definition 2.17 (Variable Assignment)
*Let* $\mathcal{I}$ *be an F-structure. A* variable assignment $\nu$ *is a mapping from the set of variables* $\mathcal{V}$ *to the domain* $U$, $\nu\colon \mathcal{V} \mapsto U$. *Variable assignments extend to id-terms in the usual way:*

- $\nu(d) := \mathcal{I}_{\mathcal{F}}(d)$ *if* $d \in \mathcal{F}$ *has arity* 0, *and,*
- *recursively,* $\nu(f(\ldots,T,\ldots)) := \mathcal{I}_{\mathcal{F}}(f)(\ldots,\nu(T),\ldots)$.

The satisfaction of an F-formula by F-structures is defined inductively over the structure of the formula. Therefore we define first what is meant by the satisfaction of F-molecules. An is-a assertion $C :: D$ or $O : C$ is true if the corresponding objects $\nu(C)$, $\nu(D)$ and $\nu(O)$ are properly related via $\prec_U$ and $\in_U$. An object molecule $O[\cdots]$ is satisfied by an F-structure $\mathcal{I}$ with respect to a variable assignment $\nu$, if the corresponding object $\nu(O)$ in $\mathcal{I}$ has properties it says it has. A P-molecule $p(T_1,\ldots,T_n)$ is satisfied if the tuple of interpreted id-terms $\langle \nu(T_1),\ldots,\nu(T_n)\rangle$ is an element of the interpretation of predicate symbol $p$.

Definition 2.18 (Satisfaction of F-molecules)
*Let* $\mathcal{I}$ *be an F-structure,* $\nu$ *be a variable assignment, and* $G$ *be an F-molecule. The F-molecule* $G$ *is satisfied by the F-structure* $\mathcal{I}$ *with respect to the variable assignment* $\nu$, *denoted* $\mathcal{I} \models_{\nu} G$ *iff all of the following holds:*

1. *When* $G$ *is an is-a assertion then:*
   (a) $\nu(C) \preceq_U \nu(D)$, *if* $G \equiv C :: D$, *or*
   (b) $\nu(O) \in_U \nu(C)$, *if* $G \equiv O : C$.

2. *When $G$ is an object molecule of the form $O$[method expressions], then for every method expression $E$ in $G$, the following conditions must hold:*

   (a) *If $E$ is a scalar data expression of the form $M$ @ $Q_1, \ldots, Q_k \rightarrow T$, the element $\mathcal{I}^{(k)}_{\rightarrow}(\nu(M))(\nu(O), \nu(Q_1), \ldots, \nu(Q_k))$*[4] *must be defined and equal $\nu(T)$.*

   (b) *If $E$ is a set-valued data expression, $M'$ @ $R_1, \ldots, R_l \twoheadrightarrow \{S_1, \ldots, S_m\}$, the set $\mathcal{I}^{(l)}_{\twoheadrightarrow}(\nu(M'))(\nu(O), \nu(R_1), \ldots, \nu(R_l))$ must be defined and contain the set $\{\nu(S_1), \ldots, \nu(S_m)\}$.*

   (c) *If $E$ is a scalar signature expression, $N$ @ $V_1, \ldots, V_n \Rightarrow (A_1, \ldots, A_r)$, then the set $\mathcal{I}^{(n)}_{\Rightarrow}(\nu(N))(\nu(O), \nu(V_1), \ldots, \nu(V_n))$ must be defined and contain the set $\{\nu(A_1), \ldots, \nu(A_r)\}$.*

   (d) *If $E$ is a set-valued signature expression, $N'$ @ $W_1, \ldots, W_s \Rrightarrow (B_1, \ldots, B_t)$, then the set $\mathcal{I}^{(s)}_{\Rrightarrow}(\nu(N'))(\nu(O), \nu(W_1), \ldots, \nu(W_s))$ must be defined and contain the set $\{\nu(B_1), \ldots, \nu(B_t)\}$.*

3. *When $G$ is a predicate molecule of the form $p(T_1, \ldots, T_n)$:*
   $\langle \nu(T_1), \ldots, \nu(T_n) \rangle \in \mathcal{I}_{\mathcal{P}}(p)$.

Instead of saying $G$ is satisfied by $\mathcal{I}$ with respect to the variable assignment $\nu$, we often say $G$ is true under $\mathcal{I}$ with respect to $\nu$.

The satisfaction of an ordinary F-formula is now defined in the standard way.

Definition 2.19 (Satisfaction of F-formulae)
*Let $\mathcal{I}$ be an F-structure, $\nu$ be a variable assignment, and $\gamma$ be an F-formula. The formula $\gamma$ is satisfied by the F-structure $\mathcal{I}$ with respect to the variable assignment $\nu$, denoted $\mathcal{I} \models_\nu \gamma$, iff all the following holds:*

- $\mathcal{I} \models_\nu \alpha$ *or* $\mathcal{I} \models_\nu \beta$, *if* $\gamma \equiv \alpha \vee \beta$,
- $\mathcal{I} \models_\nu \alpha$ *and* $\mathcal{I} \models_\nu \beta$, *if* $\gamma \equiv \alpha \wedge \beta$,
- $\mathcal{I} \not\models_\nu \alpha$ , *if* $\gamma \equiv \neg\alpha$,
- $\mathcal{I} \models_\mu \alpha$ *for every variable assignment $\mu$ that agrees with $\nu$ everywhere except possibly on $X$, if* $\gamma \equiv (\forall X)\alpha$,
- $\mathcal{I} \models_\mu \alpha$ *for some variable assignment $\mu$ that agrees with $\nu$ everywhere except possibly on $X$, if* $\gamma \equiv (\exists X)\alpha$, *or*
- $\mathcal{I} \models_\mu \alpha$ *for one and only one variable assignment $\mu$ that agrees with $\nu$ everywhere except possibly on $X$, if* $\gamma \equiv (\exists^{=1} X)\alpha$.

Definition 2.20 (Model)
*Let $\mathcal{I}$ be an F-structure, $\alpha$ be an F-formula, and $\Gamma$ be a set of F-formulae.*

[4]The expression $\mathcal{I}^{(k)}_{\rightarrow}(\nu(M))$ denotes the $k$'s component in the Cartesian product $\mathcal{I}_{\rightarrow}(\nu(M))$.

- *F-structure $\mathcal{I}$ is a model of F-formula $\alpha$, iff $\mathcal{I} \models_\nu \alpha$ for all variable assignments $\nu$.*
- *F-formula $\alpha$ is logically* implied *or* entailed *by the set $\Gamma$, iff every model of $\Gamma$ is a model of $\alpha$, written $\Gamma \models \alpha$.*

For a closed F-formula $\alpha$, a formula without free variable occurrences, we can omit the variable assignment $\nu$ and simply write $\mathcal{I} \models \alpha$, since the meaning of a closed formula is independent of the choice of variable assignments.

## 2.3 Herbrand Structures

For our purpose, the design of object-oriented databases, it is always of advantage if not a fundamental prerequisite to derive knowledge from the syntactic information provided by schemas. This is supported by so-called *Herbrand structures* as known from classical logic. There is a direct correspondence between F-structures and Herbrand structures, the details of which lie beyond the scope of this work (cf. [KLW95] for more details). Here we only present what is absolutely necessary for the understanding of our work.

Definition 2.21 (Herbrand Base)
*Let $\mathcal{L}$ be an F-logic language. The* Herbrand base *of $\mathcal{L}$, $\mathcal{HB}(\mathcal{F})$, is the set of all ground molecules.*

In classical logic a Herbrand structure is a simple subset of the Herbrand base. In F-logic similar phenomena arise as in predicate calculus with equality. Ground molecules may imply molecules in a non-trivial way. The source for this kind of implication is for example the class hierarchy. For instance, the set of is-a assertions $\{a :: b, b :: c\}$ implies the is-a assertion $a :: c$. Because of this fact a Herbrand structure in F-logic is not only a simple subset of the Herbrand base. It has to be closed under the logical implication, "$\models$".

Definition 2.22 (Herbrand Structure)
*Let $\mathcal{L}$ be an F-logic language, and $\mathbf{H} \subset \mathcal{HB}(\mathcal{F})$ be a subset of the Herbrand base. The set $\mathbf{H}$ is a Herbrand structure (H-structure for short) :iff it is closed under the logical implication, "$\models$".*

The satisfaction and logical entailment in H-structures is defined based on the membership of ground molecules, thus relying strictly on syntactic material.

Definition 2.23 (Satisfaction of F-Formulae under H-Structures)
*Let $\mathbf{H}$ be an H-structure. Then:*

- *A ground molecule $t$ is* true *in $\mathbf{H}$ (denoted $\mathbf{H} \models t$) :iff $t \in \mathbf{H}$.*
- *A ground negative literal $\neg t$ is* true *in $\mathbf{H}$, $\mathbf{H} \models \neg t$, :iff $t \notin \mathbf{H}$.*
- *A ground clause $L_1 \vee \cdots \vee L_n$ is* true *in $\mathbf{H}$ :iff at least one literal, $L_i$, is true in $\mathbf{H}$.*

- *A clause $C$ is* true *in* $\mathbf{H}$ *:iff all ground instances of $C$ are true in* $\mathbf{H}$.

*If every clause in a set of clauses $S$ is true in* $\mathbf{H}$*, we say that* $\mathbf{H}$ *is a* Herbrand model *(an* H-model*) of $S$.*

As in classical logic, it is possible to convert all formulae into prenex normal form and then to *skolemise* them. Skolemised formulae are then transformed into an equivalent clausal form. So the satisfiability of clauses is sufficient to decide on the satisfiability of general formulae.

## 2.4 Well-Typed Programmes

So far we have not defined the relationship between signature expressions and data expressions. We do not present a detailed discussion of the concepts used, because a thorough discussion is carried out in [KLW95]. Instead we give solely the rudimentary definitions necessary for our expositions, especially in Sec. 3.1.

As it is done in strongly typed languages, a method should be only invoked when a corresponding signature exists for that method. The arguments and results should then obey the types declared by the signature expressions. The method invocation must be covered by all its signature expressions.

Definition 2.24 (Typed H-Structure)
*Let* $\mathbf{H}$ *be an H-structure.*

1. *If $\alpha$ is a data atom of the form* $o[m \mathbin{@} a_1, \ldots, a_k \rightsquigarrow v] \in \mathbf{H}$[5] *and $\beta$ is a signature atom of the form* $c[m \mathbin{@} b_1, \ldots, b_k \approx\!> \ldots]$*, we shall say that $\beta$* covers *$\alpha$ if, for each* $i = 1, \ldots, k$*, we have* $o : c, a_i : b_i \in \mathbf{H}$.

2. *We shall say that* $\mathbf{H}$ *is a* typed H-structure *if the following conditions hold:*

   (a) *Every data atom in* $\mathbf{H}$ *is covered by a signature atom in* $\mathbf{H}$.

   (b) *If a data atom* $o[m \mathbin{@} a_1, \ldots, a_k \rightsquigarrow v] \in \mathbf{H}$ *is covered by a signature of the form* $c[m \mathbin{@} b_1, \ldots, b_k \approx\!> w] \in \mathbf{H}$*, then* $v : w \in \mathbf{H}$.

Based on the notation of a typed H-structure, we give a *generic* definition for *typed canonic models* [Prz88], that is models that reflect the intended semantics of a set of F-formulae; the details of these models will be immaterial for the discussions that follow.

Definition 2.25 (Typed Canonic Model)
*Let $\Gamma$ be a set of F-formulae. A* typed canonic model *of $\Gamma$ is a (usual) canonic model for $\Gamma$ that, in addition, is a typed H-structure.*

Following the generic definition of typed canonic models, we define what it means that a set of F-formulae is *well-typed.*

[5]By a misuse of notation $v$ is either a ground id-term or $\{\}$ if $\rightsquigarrow$ is $\twoheadrightarrow$.

Definition 2.26 (Well-typed Programme)
*Let $\Gamma$ be a set of F-formulae. This set, $\Gamma$, is* well-typed *if every canonic H-model of $\Gamma$ is a typed canonic H-model. Otherwise, $\Gamma$ is said to be* ill-typed.

A well-typed programme is a set of F-formulae such that the data expressions are restricted by the signature expressions. This is certainly what we want when dealing with databases. But the definitions above are sometimes not easy to work with. Therefore, we use semantic constraints in Sec. 3.1 that ensure that we work with well-typed programmes.

## 2.5 Molecule Definitions

In classical logic, we sometimes define new symbols based on other symbols [HB69, HB70, EFT92, Llo87]. These definitions do not increase the expressiveness, but often it is more convenient. We show in this section that something similar holds for F-logic as well, i. e., we can define new predicate symbols and object constructors as before and additionally we can define the interpretations of F-molecules.

In some cases (cf. Sec. 6.1) we want to *redefine* existing predicate symbols. So given a structure that already interprets the predicate symbol we want to redefine, we aim at constructing a structure that interprets this predicate symbol differently and chimes in with the original structure everywhere else. The way to do so is to rename the predicate symbol, and bestow the new symbol, in the structure at hand, with the interpretation of the original predicate symbol. Then the original predicate symbol becomes in terms of the modified structure a new, *virginal* predicate symbol and thus free to be deployed again.

Now we can encode the semantics of is-a assertions and object molecules with predicates as well. So they interpret *predefined predicate symbols*, which are never explicitly mentioned. Nevertheless we sometimes intend to change these predefined predicate symbols. As with already existing predicate symbols intended for re-interpretation, we must apply a renaming, which is harder to achieve, because their interpretation is hardwired into the definition of F-structures. To alleviate this problem, we define F-structures based on structures of the form $\mathcal{I} = (U, \mathcal{I}_{\mathcal{P}})$, which are purely *relational structures*. In doing so, we show that F-logic or better our slimmed version of F-logic can be simulated in classical logic. A fact that is claimed by the inventors of F-logic, but to our knowledge has not been shown in public.

When we define predefined predicate symbols, we have to keep in mind that restrictions are imposed on them; for example the predicate for is-a assertions has to form a partial order. These restrictions are met by means of the set $\mathbf{\Sigma}$ of sentences that confines the set of interpretations to those that comply with the restrictions.

In the following a formula $\alpha(V_1, \ldots, V_n)$ is a formula with $V_1,\ldots,V_n$ as distinct free variables. When we write $\alpha(T_1, \ldots, T_n)$ for such a formula, we denote the formula that is the result of replacing every free occurrence of a variable $V_i$ by an id-term $T_i$.

Definition 2.27 ($\mathcal{S}$-Definition)
*Let $\mathbf{\Sigma}$ be a set of classical $\mathcal{S}$-sentences.*

1. *Let $p \notin \mathcal{S}$ be a predicate symbol with arity $n$, and $\varphi_p(V_1, \ldots, V_n)$ be a classical $\mathcal{S}$-formula. We say that*

$$\forall V_1 \cdots \forall V_n (p(V_1, \ldots, V_n) \Leftrightarrow \varphi_p(V_1, \ldots, V_n))$$

*is an $\mathcal{S}$-*definition *of $p$ in $\mathbf{\Sigma}$.*

2. *Let $f \notin \mathcal{S}$ be an object constructor with arity $n$, and $\varphi_f(V_1, \ldots, V_{n+1})$ be a classical $\mathcal{S}$-formula. We say that*

$$\forall V_1 \cdots V_{n+1} (f(V_1, \ldots, V_n) \overset{\circ}{=} V_{n+1} \Leftrightarrow \varphi_f(V_1, \ldots, V_{n+1}))$$

*is an $\mathcal{S}$-*definition *of $f$ in $\mathbf{\Sigma}$, if*

$$\mathbf{\Sigma} \models \forall V_1 \cdots \forall V_n \exists^{=1} V_{n+1}\, \varphi_f(V_1, \ldots, V_{n+1}) \ .$$

*The sentence in the implication above ensures that the defining formula $\varphi_f$ acts as a total function, as it is required of the interpreting function assigned to an object constructor.*

3. *Let $\varphi_{::}(C, D)$ be a classical $\mathcal{S}$-formula. We say that*

$$\forall C \forall D (C{::}D \Leftrightarrow \varphi_{::}(C, D))$$

*is an $\mathcal{S}$-*definition *of $::$ in $\mathbf{\Sigma}$, if*

$$\mathbf{\Sigma} \models \forall C\, \varphi_{::}(C, C) \ ,$$

$$\mathbf{\Sigma} \models \forall C \forall D \forall E (\varphi_{::}(C, D) \wedge \varphi_{::}(D, E) \Rightarrow \varphi_{::}(C, E)) \ ,$$

*and*

$$\mathbf{\Sigma} \models \forall C \forall D (\varphi_{::}(C, D) \wedge \varphi_{::}(D, C) \Rightarrow C \overset{\circ}{=} D) \ .$$

*The first sentence requires that the definition of $::$ is a reflexive relation. The second requires the definition of $::$ to be a transitive relation. The third sentence guarantees that the definition of $::$ is an anti-symmetric relation. Together, this means the definition of $::$ is a partial order, as it is demanded of $\preceq_U$.*

4. *Let $\varphi_{:}(O, C)$ and $\varphi_{::}(C, D)$ be classical $\mathcal{S}$-formulae. We say that*

$$\forall O \forall C (O{:}C \Leftrightarrow \varphi_{:}(O, C))$$

*is an $\mathcal{S}$-*definition *of $:$ in $\mathbf{\Sigma}$, if*

$$\mathbf{\Sigma} \models \forall O \forall C \forall D (\varphi_{:}(O, C) \wedge \varphi_{::}(C, D) \Rightarrow \varphi_{:}(O, D)) \ .$$

*This formula captures the interplay between $:$ and $::$. So the definitions of $:$ and $::$ behave like $\in_U$ and $\preceq_U$, respectively.*

5. *Let* $\varphi_{\rightarrow_n}(M, O, A_1, \ldots, A_n, R)$ *be a classical* $\mathcal{S}$*-formula. We say that*

$$\forall M \forall O \forall A_1 \cdots \forall A_n \forall R (O[M \ @\, A_1, \ldots, A_n \rightarrow R] \Leftrightarrow \varphi_{\rightarrow_n}(M, O, A_1, \ldots, A_n, R))$$

   *is an* $\mathcal{S}$-definition *of* $\rightarrow_n$ *in* $\mathbf{\Sigma}$*, if*

$$\begin{aligned}\mathbf{\Sigma} \models\ & \forall M \forall O \forall A_1 \cdots \forall A_n \forall R \forall R' \\ & (\varphi_{\rightarrow_n}(M, O, A_1, \ldots, A_n, R) \wedge \varphi_{\rightarrow_n}(M, O, A_1, \ldots, A_n, R') \Rightarrow R \overset{\circ}{=} R') \ .\end{aligned}$$

   $\rightarrow_n$ *represents a mapping to partial functions. This property is enforced by the sentence above.*

6. *Let* $\varphi_{\sigma \twoheadrightarrow_n}(M, O, A_1, \ldots, A_n)$ *be a classical* $\mathcal{S}$*-formula. We say that*

$$\forall M \forall O \forall A_1 \cdots \forall A_n (O[M \ @\, A_1, \ldots, A_n \twoheadrightarrow \{\,\}] \Leftrightarrow \varphi_{\sigma \twoheadrightarrow_n}(M, O, A_1, \ldots, A_n))$$

   *is an* $\mathcal{S}$-definition *of* $\{\,\}_n$ *in* $\mathbf{\Sigma}$.

7. *Let* $\varphi_{\twoheadrightarrow_n}(M, O, A_1, \ldots, A_n, R)$ *and* $\varphi_{\sigma \twoheadrightarrow_n}(M, O, A_1, \ldots, A_n)$ *be classical* $\mathcal{S}$*-formulae. We say that*

$$\forall M \forall O \forall A_1 \cdots \forall A_n \forall R (O[M \ @\, A_1, \ldots, A_n \twoheadrightarrow R] \Leftrightarrow \varphi_{\twoheadrightarrow_n}(M, O, A_1, \ldots, A_n, R))$$

   *is an* $\mathcal{S}$-definition *of* $\twoheadrightarrow_n$ *in* $\mathbf{\Sigma}$*, if*

$$\begin{aligned}\mathbf{\Sigma} \models\ & \forall M \forall O \forall A_1 \cdots \forall A_n \forall R \\ & (\varphi_{\twoheadrightarrow_n}(M, O, A_1, \ldots, A_n, R) \Rightarrow \varphi_{\sigma \twoheadrightarrow_n}(M, O, A_1, \ldots, A_n)) \ .\end{aligned}$$

   *Whenever the invocation of a method on some host object yields a result, the invocation of the method is defined and yields also the empty set.*

8. *Let* $\varphi_{\sigma \approx\!> _n}(M, C, A_1, \ldots, A_n)$ *and* $\varphi_{::}(C, D)$ *be classical* $\mathcal{S}$*-formulae. We say that*

$$\forall M \forall C \forall A_1 \cdots \forall A_n (C[M \ @\, A_1, \ldots, A_n \approx\!> (\,)] \Leftrightarrow \varphi_{\sigma \approx\!> _n}(M, C, A_1, \ldots, A_n))$$

   *is an* $\mathcal{S}$-definition *of* $(\,)_{\approx\!> _n}$ *in* $\mathbf{\Sigma}$*, if*

$$\begin{aligned}\mathbf{\Sigma} \models\ & \forall M \forall C \forall A_1 \cdots \forall A_n \\ & \forall C' \forall A'_1 \cdots A'_n \\ & \quad (\varphi_{::}(C, C') \wedge \varphi_{::}(A_1, A'_1) \wedge \cdots \wedge \varphi_{::}(A_n, A'_n) \wedge \varphi_{\sigma \approx\!> _n}(M, C', A'_1, \ldots, A'_n) \\ & \quad \Rightarrow \\ & \quad \varphi_{\sigma \approx\!> _n}(M, C, A_1, \ldots, A_n)) \ .\end{aligned}$$

   *This sentence is part of the formulae that capture the anti-monotonicity of the definition of signatures.*

*9. Let* $\varphi_{\approx\!>_n}(M, C, A_1, \dots, A_n, R)$*,* $\varphi_{::}(C, D)$ *and* $\varphi_{\sigma_{\approx\!>_n}}(M, C, A_1, \dots, A_n)$ *be classical* $\mathcal{S}$*-formulae. We say that*

$$\forall M \forall C \forall A_1 \cdots \forall A_n \forall R (C[M \ @ \ A_1, \dots, A_n \approx\!> R] \Leftrightarrow \varphi_{\approx\!>_n}(M, C, A_1, \dots, A_n, R))$$

*is an* $\mathcal{S}$-definition *of* $\approx\!>_n$ *in* $\mathbf{\Sigma}$*, if*

$$\begin{aligned}\mathbf{\Sigma} \models \ & \forall M \forall C \forall A_1 \cdots \forall A_n \forall R \\ & \forall C' \forall A'_1 \cdots \forall A'_n \\ & \quad (\varphi_{::}(C, C') \wedge \varphi_{::}(A_1, A'_1) \wedge \dots \wedge \varphi_{::}(A_n, A'_n) \wedge \varphi_{\approx\!>_n}(M, C', A'_1, \dots, A'_n, R) \\ & \quad \Rightarrow \\ & \quad \varphi_{\approx\!>_n}(M, C, A_1, \dots, A_n, R)) \ ,\end{aligned}$$

$$\begin{aligned}\mathbf{\Sigma} \models \ & \forall M \forall C \forall A_1 \cdots \forall A_n \forall R \forall R' \\ & \quad (\varphi_{\approx\!>_n}(M, C, A_1, \dots, A_n, R) \wedge \varphi_{::}(R, R') \Rightarrow \varphi_{\approx\!>_n}(M, C, A_1, \dots, A_n, R'))\end{aligned}$$

*and*

$$\begin{aligned}\mathbf{\Sigma} \models \ & \forall M \forall C \forall A_1 \cdots \forall A_n \forall R \\ & \quad (\varphi_{\approx\!>_n}(M, C, A_1, \dots, A_n, R) \Rightarrow \varphi_{\sigma_{\approx\!>_n}}(M, C, A_1, \dots, A_n)) \ .\end{aligned}$$

*The first set of formulae captures the anti-monotonicity of signatures. The second requires that the set of result types is upward-closed. The third guarantees whenever a result type is given, the signature is defined.*

Throughout this work we assume that the symbols $\{::, :\} \cup \bigcup_{i \in \mathbb{N}} \{\rightarrow_i, \twoheadrightarrow_i, \{\,\}_i, \Rightarrow_i, (\,)_{\Rightarrow_i}, \Rightarrow\!\!>_i, (\,)_{\Rightarrow\!\!>_i}\}$ used in the definition above are never included in a set of symbols of an alphabet, i. e., these symbols never occur as object constructors or predicate symbols. In the following we shall assume that $\mathcal{S}$ is a set of symbols of an alphabet and $\mathbf{\Sigma}$ is a set of $\mathcal{S}$-sentences. Let $\mathcal{S}^\Delta \supset \mathcal{S}$ and $\Delta$ be a set of $\mathcal{S}$-definitions in $\mathbf{\Sigma}$ such that

- for each symbol in $(\mathcal{S}^\Delta \backslash \mathcal{S}) \cup \{::, :\} \cup \bigcup_{i \in \mathbb{N}} \{\rightarrow_i, \twoheadrightarrow_i, \{\,\}_i, \Rightarrow_i, (\,)_{\Rightarrow_i}, \Rightarrow\!\!>_i, (\,)_{\Rightarrow\!\!>_i}\}$ there is one and only one $\mathcal{S}$-definition of this symbol in $\mathbf{\Sigma}$,
- there are only $\mathcal{S}$-definitions of the symbols $(\mathcal{S}^\Delta \backslash \mathcal{S}) \cup \{::, :\} \cup \bigcup_{i \in \mathbb{N}} \{\rightarrow_i, \twoheadrightarrow_i, \{\,\}_i, \Rightarrow_i, (\,)_{\Rightarrow_i}, \Rightarrow\!\!>_i, (\,)_{\Rightarrow\!\!>_i}\}$ in $\mathbf{\Sigma}$ in the set $\Delta$.

In classical logic a structure consists only of its universe, an interpretation for function symbols and an interpretation for predicate symbols. We call such a structure $\mathcal{I} = \langle \mathcal{I}(U), \mathcal{I}(\mathcal{F}), \mathcal{I}(\mathcal{P}) \rangle$ a *classical structure.*

Lemma 2.28
*If the classical* $\mathcal{S}$*-structure* $\mathcal{I} = \langle \mathcal{I}(U), \mathcal{I}(\mathcal{F}), \mathcal{I}(\mathcal{P}) \rangle$ *is a model of* $\mathbf{\Sigma}$ *in the classical sense, then there is one and only one* $\mathcal{S}^\Delta$*-structure* $\mathcal{I}^\Delta$ *with*

$$\mathcal{I}^\Delta \lceil_{\mathcal{S}} = \mathcal{I} \text{ and } \mathcal{I}^\Delta \models \Delta \ . \tag{2.1}$$

Proof. We show the existence of such an $\mathcal{S}^\Delta$-structure

$$\mathcal{I}^\Delta = \left\langle U^\Delta, \prec_U^\Delta, \in_U^\Delta, \mathcal{I}_\mathcal{F}^\Delta, \mathcal{I}_\mathcal{P}^\Delta, \mathcal{I}_\rightarrow^\Delta, \mathcal{I}_\twoheadrightarrow^\Delta, \mathcal{I}_\Rightarrow^\Delta, \mathcal{I}_\twoheadRightarrow^\Delta \right\rangle$$

by construction. The universe $U^\Delta$ is identical to the universe $\mathcal{I}(U)$,

$$U^\Delta := \mathcal{I}(U) \ .$$

Defining the rest of the structure, we stipulate the interpretation of all symbols in $\mathcal{S}$ to be identical to the interpretation of these symbols under $\mathcal{I}$.

- $\mathcal{I}_\mathcal{F}^\Delta(f) := \mathcal{I}(\mathcal{F})(f)$ for all object constructors $f \in \mathcal{S}$.
- $\mathcal{I}_\mathcal{P}^\Delta(p) := \mathcal{I}(\mathcal{P})(p)$ for all predicate symbols $p \in \mathcal{S}$.

Then we define the interpretations of all symbols $(\mathcal{S}^\Delta \backslash \mathcal{S}) \cup \{::, :\} \cup \bigcup_{i \in \mathbb{N}} \{\rightarrow_i, \twoheadrightarrow_i, \{\,\}_i, \Rightarrow_i, (\,)_{\Rightarrow_i}, \twoheadRightarrow_i, (\,)_{\twoheadRightarrow_i}\}$ based on the truth values of the respective defining formulae under the structure $\mathcal{I}$.

- $\mathcal{I}_\mathcal{F}^\Delta(f)(\nu(V_1), \ldots, \nu(V_n)) := \nu(V_{n+1})$ :iff $\mathcal{I} \models_\nu \varphi_f(V_1, \ldots, V_{n+1})$ for all object constructors $f \in \mathcal{S}^\Delta \backslash \mathcal{S}$ with arity $n$ and variable assignments $\nu$.
- $(\nu(V_1), \ldots, \nu(V_n)) \in \mathcal{I}_\mathcal{P}^\Delta(p)$ :iff $\mathcal{I} \models_\nu \varphi_p(V_1, \ldots, V_n)$ for all predicate symbols $p \in \mathcal{S}^\Delta \backslash \mathcal{S}$ with arity $n$ and variable assignments $\nu$.
- $\nu(C) \preceq_U^\Delta \nu(D)$ :iff $\mathcal{I} \models_\nu \varphi_{::}(C, D)$ for all variable assignments $\nu$.
- $\nu(O) \in_U^\Delta \nu(C)$ :iff $\mathcal{I} \models_\nu \varphi_{:}(O, C)$ for all variable assignments $\nu$.
- $\mathcal{I}_\rightarrow^{\Delta\,(n)}(\nu(M))(\nu(O), \nu(A_1), \ldots, \nu(A_n)) \quad := \quad \nu(R)$ :iff $\mathcal{I} \models_\nu \varphi_{\rightarrow_n}(M, O, A_1, \ldots, A_n, R)$ for all $\mathcal{S}$-definitions $\rightarrow_n$ in the set of $\mathcal{S}$-definitions $\Delta$ with defining formula $\varphi_{\rightarrow_n}(M, O, A_1, \ldots, A_n)$ and all variable assignments $\nu$.
- For all $\mathcal{S}$-definitions $\twoheadrightarrow_n$ in the set $\Delta$ of $\mathcal{S}$-definitions,[6]

$$\begin{aligned} &\mathcal{I}_\twoheadrightarrow^{\Delta\,(n)}(\nu(M))(\nu(O), \nu(A_1), \ldots, \nu(A_n)) \\ &\quad := \begin{cases} \{\mu(R) \mid \text{ ex. } \mu \equiv_R \nu \colon \mathcal{I} \models_\mu \varphi_{\twoheadrightarrow_n}(M, O, A_1, \ldots, A_n, R)\} \ , \\ \quad \text{if } \mathcal{I} \models_\nu \varphi_{\sigma_{\twoheadrightarrow_n}}(M, O, A_1, \ldots, A_n) \\ \text{undefined else} \ . \end{cases} \end{aligned}$$

- For all $\mathcal{S}$-definitions $\twoheadRightarrow_n$ in the set $\Delta$ of $\mathcal{S}$-definitions,

$$\begin{aligned} &\mathcal{I}_\twoheadRightarrow^{\Delta\,(n)}(\nu(M))(\nu(C), \nu(A_1), \ldots, \nu(A_n)) \\ &\quad := \begin{cases} \{\mu(R) \mid \text{ ex. } \mu \equiv_R \nu \colon \mathcal{I} \models_\mu \varphi_{\twoheadRightarrow_n}(M, C, A_1, \ldots, A_n, R)\} \ , \\ \quad \text{if } \mathcal{I} \models_\nu \varphi_{\sigma_{\twoheadRightarrow_n}}(M, C, A_1, \ldots, A_n) \\ \text{undefined else} \ . \end{cases} \end{aligned}$$

[6]Here $\mu \equiv_R \nu$ for two variable assignments $\mu$ and $\nu$ means that the assignment $\mu$ is equal to the assignment $\nu$ except possibly for the assignment of the variable $R$.

For this construction $\mathcal{I}^\Delta$ we prove the following properties:

1. The construction $\mathcal{I}^\Delta$ is an F-structure.

   The classical structure $\mathcal{I}$ is a model of $\mathbf{\Sigma}$, which guarantees that the defining formulae behave like the corresponding elements of an F-structure.
2. $\mathcal{I}^\Delta\lceil_{\mathcal{S}} = \mathcal{I}$.

   Trivial according to the construction of $\mathcal{I}^\Delta$.
3. $\mathcal{I}^\Delta \models \Delta$.

   Trivial according to the construction of $\mathcal{I}^\Delta$.

It remains to show that the constructed structure $\mathcal{I}^\Delta$ is unique. So let $\mathcal{J}$ be an $\mathcal{S}^\Delta$-structure with $\mathcal{J} \models \Delta$. Then for every newly defined symbol $\sigma \in (\mathcal{S}^\Delta \backslash \mathcal{S}) \cup \{::, :\} \cup \bigcup_{i\in\mathbb{N}}\{\rightarrow_i, \twoheadrightarrow_i, \{\,\}_i, \Rightarrow_i, (\,)_{\Rightarrow_i}, \Twoheadrightarrow_i, (\,)_{\Twoheadrightarrow_i}\}$ the defining formula $\varphi_\sigma$ uniquely determines the interpretation of $\sigma$ in the structure $\mathcal{J}$, which is consequently equal to $\mathcal{I}^\Delta$. □

A spin-off of the preceding lemma is that we know that we can define some F-structures by means of classical structures. We waive the question whether all F-structures can be defined by $\mathcal{S}$-definitions until later in this chapter.

It should be intuitively clear that the introduction of newly defined symbols does not increase the expressiveness, because it is possible to find for every formula $\alpha$ containing defined symbols an equivalent formula $\alpha^\nabla$ without defined symbols. This is achieved by replacing every defined symbol by its defining formula. The following theorem captures this formally.

THEOREM 2.29 (DEFINITION ENLARGEMENT)
*Let $\mathcal{S}^\Delta \supset \mathcal{S}$ be a set of symbols, and $\mathbf{\Sigma}$ be a set of classical $\mathcal{S}$-sentences. Every symbol in $\mathcal{S}^\Delta \backslash \mathcal{S} \cup \{::, :\} \cup \bigcup_{i\in\mathbb{N}}\{\rightarrow_i, \twoheadrightarrow_i, \{\,\}_i, \Rightarrow_i, (\,)_{\Rightarrow_i}, \Twoheadrightarrow_i, (\,)_{\Twoheadrightarrow_i}\}$ has exactly one $\mathcal{S}$-definition, and $\Delta$ be the set of these $\mathcal{S}$-definitions. Then there exists for every $\mathcal{S}^\Delta$-formula $\alpha(V_1, \ldots, V_n)$ a classical $\mathcal{S}$-formula $\alpha^\nabla(V_1, \ldots, V_n)$ such that:*

1. *If $\mathcal{I}$ is a classical $\mathcal{S}$-structure with $\mathcal{I} \models \mathbf{\Sigma}$ and $u_1, \ldots, u_n \in \mathcal{I}(U)$, then*

   $$\mathcal{I}^\Delta \models_\nu \alpha \text{ iff } \mathcal{I} \models_\nu \alpha^\nabla \ ,$$

   *where $\nu(V_i) = u_i$. (The structure $\mathcal{I}^\Delta$ be the uniquely determined $\mathcal{S}^\Delta$-enlargement of $\mathcal{I}$ with $\mathcal{I}^\Delta \models \Delta$, according to Lem. 2.28.)*

2. $\mathbf{\Sigma} \cup \Delta \models \alpha \Leftrightarrow \alpha^\nabla$.

PROOF FOR 1. We define $^\nabla : \mathcal{L}^{\mathcal{S}^\Delta} \rightarrow \mathcal{L}^{\mathcal{S}}$ inductively. We denote a language $\mathcal{L}$ over an alphabet $\mathcal{A}$ as $\mathcal{L}^{\mathcal{S}}$ where $\mathcal{S}$ is the set of symbols of the alphabet $\mathcal{A}$. For $\alpha \in \mathcal{L}^{\mathcal{S}^\Delta}$ let $V_1, \ldots$ be an enumeration of all variables occurring in $\alpha$. Then $V_1', \ldots$ is an enumeration of variables not occurring in $\alpha$.

We first define $\alpha^\nabla$ for expressions $\alpha$ of the form $t \overset{\circ}{=} X$, where $X$ is a variable, inductively over $t$.

$t \equiv Y$ with $Y$ is a variable:

$$[Y \overset{\circ}{=} X]^{\nabla} := Y \overset{\circ}{=} X \ .$$

$t \equiv f(t_1, \ldots, t_n)$:

$$[f(t_1,\ldots,t_n) \overset{\circ}{=} X]^{\nabla} := \begin{cases} \exists V_1' \cdots \exists V_n' \Big( [t_1 \overset{\circ}{=} V_1']^{\nabla} \wedge \ldots \wedge [t_n \overset{\circ}{=} V_n']^{\nabla} \wedge \\ \varphi_f(V_1', \ldots, V_n', X) \Big) \ , \\ \hfill \text{if } f \in \mathcal{S}^{\Delta} \backslash \mathcal{S} \ , \\ \exists V_1' \cdots \exists V_n' \Big( [t_1 \overset{\circ}{=} V_1']^{\nabla} \wedge \ldots \wedge [t_n \overset{\circ}{=} V_n']^{\nabla} \wedge \\ f(V_1', \ldots, V_n') \overset{\circ}{=} X \Big) \ , \\ \hfill \text{if } f \in \mathcal{S} \ . \end{cases}$$

We define $\alpha^{\nabla}$ for remaining $\mathcal{S}^{\Delta}$-formulae $\alpha(V_1, \ldots, V_n)$ by induction on the structure of $\alpha$. Without loss of generality we assume that all molecules are broken apart into their constituent atoms.

1. $\alpha \equiv t_1 \overset{\circ}{=} t_2$ where $t_2$ is no variable:

$$\alpha^{\nabla} := \exists V_1' ([t_1 \overset{\circ}{=} V_1']^{\nabla} \wedge [t_2 \overset{\circ}{=} V_1']^{\nabla})$$

2. $\alpha \equiv p(t_1, \ldots, t_n)$:

$$\alpha^{\nabla} := \begin{cases} \exists V_1' \cdots \exists V_n' \Big( [t_1 \overset{\circ}{=} V_1']^{\nabla} \wedge \ldots \wedge [t_n \overset{\circ}{=} V_n']^{\nabla} \wedge \\ \varphi_p(V_1', \ldots, V_n') \Big) \ , \\ \hfill \text{if } p \in \mathcal{S}^{\Delta} \backslash \mathcal{S} \ , \\ \exists V_1' \cdots \exists V_n' \Big( [t_1 \overset{\circ}{=} V_1']^{\nabla} \wedge \ldots \wedge [t_n \overset{\circ}{=} V_n']^{\nabla} \wedge \\ p(V_1', \ldots, V_n') \Big) \ , \\ \hfill \text{if } p \in \mathcal{S} \end{cases}$$

3. $\alpha \equiv t_1{::}t_2$:

$$\alpha^{\nabla} := \ \exists V_1' \exists V_2' \Big( [t_1 \overset{\circ}{=} V_1']^{\nabla} \wedge [t_2 \overset{\circ}{=} V_2']^{\nabla} \wedge \varphi_{::}(V_1', V_2') \Big)$$

4. $\alpha \equiv t_1{:}t_2$:

$$\alpha^{\nabla} := \ \exists V_1' \exists V_2' \Big( [t_1 \overset{\circ}{=} V_1']^{\nabla} \wedge [t_2 \overset{\circ}{=} V_2']^{\nabla} \wedge \varphi_{:}(V_1', V_2') \Big)$$

5. $\alpha \equiv t_o[\,]$:

$$\alpha^{\nabla} := \text{true}$$

6. $\alpha \equiv t_o[t_m \mathbin{@} t_{a_1}, \ldots, t_{a_n} \blacktriangleright t_r]$:

$$\begin{aligned}\alpha^\nabla := \exists V'_1 \cdots \exists V'_{n+3}\Big( & [t_{a_1} \overset{\circ}{=} V'_1]^\nabla \wedge \ldots \wedge [t_{a_n} \overset{\circ}{=} V'_n]^\nabla \wedge \\ & [t_m \overset{\circ}{=} V'_{n+1}]^\nabla \wedge [t_o \overset{\circ}{=} V'_{n+2}]^\nabla \wedge [t_r \overset{\circ}{=} V'_{n+3}]^\nabla \wedge \\ & \varphi_{\blacktriangleright n}(V'_{n+1}, V'_{n+2}, V'_1, \ldots, V'_n, V'_{n+3})\Big)\end{aligned}$$

Here $\blacktriangleright$ stands for $\rightarrow$, $\twoheadrightarrow$, $\Rightarrow$ or $\Rrightarrow$.

7. $\alpha \equiv t_o[t_m \mathbin{@} t_{a_1}, \ldots, t_{a_n} \twoheadrightarrow \{\,\}]$:

$$\begin{aligned}\alpha^\nabla := \exists V'_1 \cdots \exists V'_{n+2}\Big( & [t_{a_1} \overset{\circ}{=} V'_1]^\nabla \wedge \ldots \wedge [t_{a_n} \overset{\circ}{=} V'_n]^\nabla \wedge \\ & [t_m \overset{\circ}{=} V'_{n+1}]^\nabla \wedge [t_o \overset{\circ}{=} V'_{n+2}]^\nabla \wedge \\ & \varphi_{\sigma \twoheadrightarrow n}(V'_{n+1}, V'_{n+2}, V'_1, \ldots, V'_n)\Big)\end{aligned}$$

8. $\alpha \equiv t_c[t_m \mathbin{@} t_{a_1}, \ldots, t_{a_n} \Rrightarrow (\,)]$:

$$\begin{aligned}\alpha^\nabla := \exists V'_1 \cdots \exists V'_{n+2}\Big( & [t_{a_1} \overset{\circ}{=} V'_1]^\nabla \wedge \ldots \wedge [t_{a_n} \overset{\circ}{=} V'_n]^\nabla \wedge \\ & [t_m \overset{\circ}{=} V'_{n+1}]^\nabla \wedge [t_c \overset{\circ}{=} V'_{n+2}]^\nabla \wedge \\ & \varphi_{\sigma \Rrightarrow n}(V'_{n+1}, V'_{n+2}, V'_1, \ldots, V'_n)\Big)\end{aligned}$$

9. $\alpha \equiv \neg\beta$:
$$\alpha^\nabla := \neg(\beta)^\nabla$$

10. $\alpha \equiv (\beta_1 \vee \beta_2)$:
$$\alpha^\nabla := (\beta_1^\nabla \vee \beta_2^\nabla)$$

11. $\alpha \equiv \exists V \beta$:
$$\alpha^\nabla := \exists V(\beta)^\nabla$$

12. $\alpha \equiv \exists^{=1} V \beta$:
$$\alpha^\nabla := \exists V^{=1}(\beta)^\nabla$$

13. $\alpha \equiv \forall V \beta$:
$$\alpha^\nabla := \forall V(\beta)^\nabla$$

It is not difficult to prove 1 by the help of the definitions above. The proof is carried out inductively. In particular, the following holds:

$$\mathcal{I}^\Delta \models_\nu (\alpha(V_1, \ldots, V_n) \Leftrightarrow \alpha^\nabla(V_1, \ldots, V_n)) \ , \tag{2.2}$$

where $\nu(V_i) = u_i$.

Proof for 2. Let $\mathcal{I}'$ be an $\mathcal{S}^\Delta$-structure with $\mathcal{I}' \models \mathbf{\Sigma} \cup \Delta$ and $u_1, \ldots, u_n \in \mathcal{I}'(U)$. It remains to show that $\mathcal{I}' \models_\nu (\alpha(V_1, \ldots, V_n) \Leftrightarrow \alpha^\nabla(V_1, \ldots, V_n))$ where $\nu(V_i) = u_i$ for $i \in \{1, \ldots, n\}$. We know that $\mathcal{I}^\Delta = \mathcal{I}'$ for the structure $\mathcal{I} := \mathcal{I}'\lceil_{\mathcal{S}}$ by Lem. 2.28. Therefore, the assertion follows from (2.2).

□

A symbol set is called *relational* if it contains only predicate symbols. In classical logic function symbols can be replaced by predicate symbols to obtain a relational symbol set. The idea is to look at the graph of a function instead of the function itself. This idea can be extended to F-logic. In the case of the restricted F-logic we use, we can even encode is-a assertions and object molecules by means of predicate symbols. This is not surprising since F-logic is a first-order logic.

Definition 2.30
*Let $\mathcal{S}$ be a set of symbols. $p_f$ be a new predicate symbol with arity $n+1$ for every object constructor $f \in \mathcal{S}$ with arity $n$. $\mathcal{S}^r$ consists of the predicate symbols in $\mathcal{S}$, the newly introduced predicate symbols for object constructors, and the predicate symbols*[7] $::^2$, $:^2$, $\{\rightarrow_i^{i+3} \mid i \in \mathbb{N}\}$, $\{\twoheadrightarrow_i^{i+3} \mid i \in \mathbb{N}\}$, $\{\sigma_{\twoheadrightarrow_i}^{i+2} \mid i \in \mathbb{N}\}$, $\{\Rightarrow_i^{i+3} \mid i \in \mathbb{N}\}$, $\{\sigma_{\Rightarrow_i}^{i+2} \mid i \in \mathbb{N}\}$, $\{\Rrightarrow_i^{i+3} \mid i \in \mathbb{N}\}$, *and* $\{\sigma_{\Rrightarrow_i}^{i+2} \mid i \in \mathbb{N}\}$.

The additional predicate symbols are meant to encode is-a assertions and object molecules. We deliberately choose predicate symbols close to the "built-in" predicates. Thus it should be easy for the reader to identify their counterparts. This is different for the predicates like $\sigma_{\twoheadrightarrow_i}$. Their meaning is not self-explanatory. They are necessary to indicate that a value of a set-valued method or of a signature is defined at all. Without them the distinction between the case that the invocation of a method on a host object is undefined for given arguments or is the empty set is not possible.

We relate each $\mathcal{S}$-structure $\mathcal{I}$ to an $\mathcal{S}^r$-structure $\mathcal{I}^r$, by replacing functions by their graphs and the "built-in" predicates by ordinary predicates.

Definition 2.31
*Let $\mathcal{I}$ be an $\mathcal{S}$-structure. The structure $\mathcal{I}^r := \langle U^r, \mathcal{I}^r_{\mathcal{P}} \rangle$ is the $\mathcal{S}^r$-structure defined as follows:*

1. $U^r := U$;

2. *for every predicate symbol* $p \in \mathcal{S}$: $\mathcal{I}^r_{\mathcal{P}}(p) := \mathcal{I}_{\mathcal{P}}(p)$;

3. *for every object constructor* $f \in \mathcal{S}$ *with arity* $n$:

$$(u_1, \ldots, u_{n+1}) \in \mathcal{I}^r_{\mathcal{P}}(p_f) \text{ :iff } \mathcal{I}_{\mathcal{F}}(f)(u_1, \ldots, u_n) = u_{n+1} \;;$$

4. *for* $::$:

$$(u_1, u_2) \in \mathcal{I}^r_{\mathcal{P}}(::) \text{ :iff } u_1 \preceq_U u_2 \;;$$

5. *for* $:$:

$$(u_1, u_2) \in \mathcal{I}^r_{\mathcal{P}}(:) \text{ :iff } u_1 \in_U u_2 \;;$$

6. *for* $\rightarrow_n$:

$$(u_m, u_o, a_1, \ldots, a_n, u) \in \mathcal{I}^r_{\mathcal{P}}(\rightarrow_n) \text{ :iff } \mathcal{I}^{(n)}_{\rightarrow}(u_m)(u_o, a_1, \ldots, a_n) = u \;;$$

[7] We indicate the arity of these new predicate symbols by the superscript numbers.

7. *for* $\twoheadrightarrow_n$*:*

$$(u_m, u_o, a_1, \ldots, a_n, u) \in \mathcal{I}^r_{\mathcal{P}}(\twoheadrightarrow_n) \text{ :iff } u \in \mathcal{I}^{(n)}_{\twoheadrightarrow}(u_m)(u_o, a_1, \ldots, a_n) \ ;$$

8. *for* $\sigma_{\twoheadrightarrow_n}$*:*

$$(u_m, u_o, a_1, \ldots, a_n) \in \mathcal{I}^r_{\mathcal{P}}(\sigma_{\twoheadrightarrow_n}) \text{ :iff } \mathcal{I}^{(n)}_{\twoheadrightarrow}(u_m)(u_o, a_1, \ldots, a_n) \text{ defined } ;$$

9. *for* $\gtrapprox_n$*:*

$$(u_m, u_c, a_1, \ldots, a_n, u) \in \mathcal{I}^r_{\mathcal{P}}(\gtrapprox_n) \text{ :iff } u \in \mathcal{I}^{(n)}_{\gtrapprox}(u_m)(u_c, a_1, \ldots, a_n) \ ;$$

10. *for* $\sigma_{\gtrapprox_n}$*:*

$$(u_m, u_c, a_1, \ldots, a_n) \in \mathcal{I}^r_{\mathcal{P}}(\sigma_{\gtrapprox_n}) \text{ :iff } \mathcal{I}^{(n)}_{\gtrapprox}(u_m)(u_c, a_1, \ldots, a_n) \text{ defined } .$$

Accordingly, it is possible to translate every $\mathcal{S}$-formula into an $\mathcal{S}^r$-formula by replacing atomar sub-formulae like $f(X, Y) \overset{\circ}{=} Z$ by $p_f(X, Y, Z)$. Nested formulae are treated in the same manner as in the proof of Theor. 2.29. A sentence $\alpha$ is satisfied by an $\mathcal{S}$-structure $\mathcal{I}$ if and only if $\alpha^r$ is satisfied by the $\mathcal{S}^r$-structure $\mathcal{I}^r$, as proven by the next theorem. It contains the formal definition of $\alpha^r$.

Theorem 2.32

*For every $\mathcal{S}$-formula $\alpha(V_1, \ldots, V_n)$ there exists a classical $\mathcal{S}^r$-formula $\alpha^r(V_1, \ldots, V_n)$ that does not contain object constructors, such that for all $\mathcal{S}$-structures $\mathcal{I}$ and all $u_1, \ldots, u_n \in \mathcal{I}(U)$:*

$$\mathcal{I} \models_\nu \alpha \text{ iff } \mathcal{I}^r \models_\nu \alpha^r \ , \tag{2.3}$$

*where $\nu(V_i) = u_i$ for $i \in \{1, \ldots, n\}$.*

Proof. Instead of showing this assertion directly, we can make use of Theor. 2.29. We start by obtaining the relational structure $\mathcal{I}^r$ from the structure $\mathcal{I}$. Afterwards we define the set $\Delta$ of $\mathcal{S}$-definitions in $\Sigma$, also defined by us in this proof, such that $\mathcal{I}^{r\Delta}\lceil_{\mathcal{S}} = \mathcal{I}$, which is proven after the definitions of the sets $\Sigma$ and $\Delta$. The set of symbols $\mathcal{S} \cup \mathcal{S}^r$ takes on the rôle of $\mathcal{S}^\Delta$ and $\mathcal{S}^r$ the rôle of $\mathcal{S}$ according to Theor. 2.29.

The purpose of the set of sentences $\mathbf{\Sigma}$ is to guarantee that the predicates used to encode functions and the special "built-in" predicates behave *properly.* The rules of conduct are set up by the definition of F-structures.

We give for every special predicate a set of sentences and explain their meaning.

$$\begin{aligned}\mathbf{\Sigma}_{::} := \ & \{\forall V {::}(V, V)\} \\ & \cup \\ & \{\forall V_x \forall V_y \forall V_z ({::}(V_x, V_y) \wedge {::}(V_y, V_z) \Rightarrow {::}(V_x, V_z))\} \\ & \cup \\ & \{\forall V_c \forall V_d ({::}(V_c, V_d) \wedge {::}(V_d, V_c) \Rightarrow V_c \overset{\circ}{=} V_d)\}\end{aligned}$$

The first sentence requires that the interpretation of :: is a reflexive relation. The second requires the interpretation of :: to be a transitive relation. The third sentence guarantees that the definition of :: is an anti-symmetric relation. Together, this means the interpretation of :: is a partial order, as it is demanded of $\preceq_U$.

$$\mathbf{\Sigma}_{:} := \{\forall V_u \forall V_v \forall V_w (:(V_u, V_v) \land ::(V_v, V_w) \Rightarrow :(V_u, V_w))\}$$

This formula captures the interplay between : and ::. The interpretations of : and :: behave like $\in_U$ and $\preceq_U$, respectively.

$$\mathbf{\Sigma}_{\mathcal{F}} := \{\forall V_1 \cdots \forall V_n \exists^{=1} V_{n+1}\, p_f(V_1, \ldots, V_{n+1}) \mid f \in \mathcal{S}\}$$

It ensures that the interpretation of $p_f$ is a graph of a function.

$$\begin{aligned}\mathbf{\Sigma}_{\rightarrow} := \{&\forall V_m \forall V_o \forall V_{a_1} \cdots \forall V_{a_n} \forall V_r \forall V_{r'} \\ &\quad \rightarrow_n(V_m, V_o, V_{a_1}, \ldots, V_{a_n}, V_r) \land \rightarrow_n(V_m, V_o, V_{a_1}, \ldots, V_{a_n}, V_{r'}) \Rightarrow V_r \overset{\circ}{=} V_{r'} \\ &\qquad \mid n \in \mathbb{N}\}\end{aligned}$$

$\rightarrow_n$ represents a mapping to partial functions. This property is enforced by the above sentences.

$$\begin{aligned}\mathbf{\Sigma}_{\twoheadrightarrow} := \{&\forall V_m \forall V_o \forall V_{a_1} \cdots \forall V_{a_n} \forall V_r \\ &\quad \twoheadrightarrow_n(V_m, V_o, V_{a_1}, \ldots, V_{a_n}, V_r) \Rightarrow \sigma_{\twoheadrightarrow_n}(V_m, V_o, V_{a_1}, \ldots, V_{a_n}) \\ &\qquad \mid n \in \mathbb{N}\}\end{aligned}$$

Whenever the invocation of a method on some host object gives a result, the invocation of the method is defined. This has to be reflected by the predicates $\sigma_{\twoheadrightarrow_n}$.

$$\begin{aligned}\mathbf{\Sigma}_{\Rrightarrow} := \ &\{\forall V_m \forall V_c \forall V_{a_1} \cdots \forall V_{a_n} \forall V_r \\ &\quad \forall V_c' \forall V_{a_1}' \cdots \forall V_{a_n}' \\ &\qquad ::(V_c, V_c') \land ::(V_{a_1}, V_{a_1}') \land \ldots \land ::(V_{a_n}, V_{a_n}') \land \\ &\qquad \Rrightarrow_n(V_m, V_c', V_{a_1}', \ldots, V_{a_n}', V_r) \Rightarrow \Rrightarrow_n(V_m, V_c, V_{a_1}, \ldots, V_{a_n}, V_r) \mid n \in \mathbb{N}\} \\ &\cup \\ &\{\forall V_m \forall V_c \forall V_{a_1} \cdots \forall V_{a_n} \forall V_r \forall V_r' \\ &\qquad \Rrightarrow_n(V_m, V_c, V_{a_1}, \ldots, V_{a_n}, V_r) \land ::(V_r, V_r') \Rightarrow \Rrightarrow_n(V_m, V_c, V_{a_1}, \ldots, V_{a_n}, V_r') \\ &\qquad\quad \mid n \in \mathbb{N}\} \\ &\cup \\ &\{\forall V_m \forall V_c \forall V_{a_1} \cdots \forall V_{a_n} \forall V_r \\ &\qquad \Rrightarrow_n(V_m, V_c, V_{a_1}, \ldots, V_{a_n}, V_r) \Rightarrow \sigma_{\Rrightarrow_n}(V_m, V_c, V_{a_1}, \ldots, V_{a_n}) \\ &\qquad\quad \mid n \in \mathbb{N}\}\end{aligned}$$

The first set of formulae captures the anti-monotonicity of signatures. The second requires that the set of result types is upward-closed. The third guarantees whenever a result type is given, the signature is defined.

$$\begin{aligned}\mathbf{\Sigma}_{\sigma_{\Rrightarrow}} := \{&\forall V_m \forall V_c \forall V_{a_1} \cdots \forall V_{a_n} \\ &\quad \forall V_c' \forall V_{a_1}' \cdots \forall V_{a_n}' \\ &\qquad ::(V_c, V_c') \land ::(V_{a_1}, V_{a_1}') \land \ldots \land ::(V_{a_n}, V_{a_n}') \land \\ &\qquad \sigma_{\Rrightarrow_n}(V_m, V_c', V_{a_1}', \ldots, V_{a_n}') \Rightarrow \sigma_{\Rrightarrow_n}(V_m, V_c, V_{a_1}, \ldots, V_{a_n}) \mid n \in \mathbb{N}\}\end{aligned}$$

This sentence is still part of the formulae that capture the anti-monotonicity of the definition of signatures.

We define

$$\boldsymbol{\Sigma} := \boldsymbol{\Sigma}_{::} \cup \boldsymbol{\Sigma}_{:} \cup \boldsymbol{\Sigma}_{\mathcal{F}} \cup \boldsymbol{\Sigma}_{\rightarrow} \cup \boldsymbol{\Sigma}_{\twoheadrightarrow} \cup \boldsymbol{\Sigma}_{\approx\!>} \cup \boldsymbol{\Sigma}_{\sigma_{\approx\!>}}$$

and

$$\begin{array}{ll}
\Delta := & \{\forall C \forall D (C{::}D \Leftrightarrow {::}(C, D))\} \\
& \cup \\
& \{\forall O \forall C (O{:}C \Leftrightarrow {:}(O, C))\} \\
& \cup \\
& \{\forall V_1 \cdots \forall V_{n+1}\; f(V_1, \ldots, V_n) \overset{\circ}{=} V_{n+1} \Leftrightarrow p_f(V_1, \ldots, V_{n+1}) \mid f \in \mathcal{S} \text{ with arity } n\} \\
& \cup \\
& \{\forall M \forall O \forall A_1 \cdots \forall A_n \forall R \\
& \quad O[M \text{ @ } A_1, \ldots, A_n \rightsquigarrow R] \Leftrightarrow {\rightsquigarrow}_n(M, O, A_1, \ldots, A_n, R) \mid n \in \mathbb{N}\} \\
& \cup \\
& \{\forall M \forall O \forall A_1 \cdots \forall A_n \\
& \quad O[M \text{ @ } A_1, \ldots, A_n \twoheadrightarrow \{\,\}] \Leftrightarrow \sigma_{\twoheadrightarrow_n}(M, O, A_1, \ldots, A_n) \mid n \in \mathbb{N}\} \\
& \cup \\
& \{\forall M \forall C \forall A_1 \cdots \forall A_n \forall R \\
& \quad C[M \text{ @ } A_1, \ldots, A_n \approx\!> R] \Leftrightarrow {\approx\!>}_n(M, C, A_1, \ldots, A_n, R) \mid n \in \mathbb{N}\} \\
& \cup \\
& \{\forall M \forall C \forall A_1 \cdots \forall A_n \\
& \quad C[M \text{ @ } A_1, \ldots, A_n \approx\!> (\,)] \Leftrightarrow \sigma_{\approx\!>_n}(M, C, A_1, \ldots, A_n) \mid n \in \mathbb{N}\}\ .
\end{array}$$

The set $\Delta$ contains for every object constructor $f$ and every "built-in" predicate one $\mathcal{S}^r$-definition in $\boldsymbol{\Sigma}$.

Because of Theor. 2.29 it holds for every $\mathcal{S} \cup \mathcal{S}^r$-formula $\alpha(V_1, \ldots, V_n)$ that $\alpha^{\nabla}(V_1, \ldots, V_n)$ is a classical $\mathcal{S}^r$-formula not containing object constructors and that every $\mathcal{S}^r$-structure $\mathcal{I}$ satisfying $\boldsymbol{\Sigma}$ and for all $u_1, \ldots, u_n \in \mathcal{I}(U)$:

$$\mathcal{I}^{\Delta} \models_{\nu} \alpha \text{ iff } \mathcal{I} \models_{\nu} \alpha^{\nabla}\ , \tag{2.4}$$

where $\nu(V_i) = u_i$.

For an $\mathcal{S}$-formula $\alpha$ we define $\alpha^r := \alpha^{\nabla}$.

It remains to show that (2.3) is true. Let $\mathcal{I}$ be an $\mathcal{S}$-structure. Because of its definition $\mathcal{I}^r$ satisfies $\boldsymbol{\Sigma}$, therefore $\mathcal{I}^{r\Delta} \models \Delta$. Because of $\mathcal{I}^{r\Delta} \models \Delta$ exactly those interpretations for predicate symbols, object constructors and special "built-in" predicate symbols are added to $\mathcal{I}^r$ in order to obtain $\mathcal{I}^{r\Delta}$ that were eliminated in the transformation from $\mathcal{I}$ to $\mathcal{I}^r$, hence $\mathcal{I}^{r\Delta}\restriction_{\mathcal{S}} = \mathcal{I}$. Let $\alpha(V_1, \ldots, V_n)$ be an $\mathcal{S}$-formula and $\nu$ be a variable assignment:

$$\begin{array}{rlr}
\mathcal{I} \models_{\nu} \alpha & \text{iff } \mathcal{I}^{r\Delta}\restriction_{\mathcal{S}} \models_{\nu} \alpha & \\
& \text{iff } \mathcal{I}^{r\Delta} \models_{\nu} \alpha & \\
& \text{iff } \mathcal{I}^{r} \models_{\nu} \alpha^{\nabla} & (2.4) \\
& \text{iff } \mathcal{I}^{r} \models_{\nu} \alpha^{r} & (\alpha^r = \alpha^{\nabla})\ .
\end{array}$$

□

The converse of the above theorem is also true. It is proven similarly.

THEOREM 2.33
*For every classical $\mathcal{S}^r$-formula $\alpha(V_1, \ldots, V_n)$ there exists an $\mathcal{S}$-formula $\alpha^{-r}$, such that for all $\mathcal{S}$-structures $\mathcal{I}$ and all $u_1, \ldots, u_n \in \mathcal{I}(U)$:*

$$\mathcal{I}^r \models_\nu \alpha \textit{ iff } \mathcal{I} \models_\nu \alpha^{-r} \quad , \tag{2.5}$$

*where $\nu(V_i) = u_i$ for $i \in \{1, \ldots, n\}$.*

PROOF. The proof is similar to the proof of Theor. 2.32. We stipulate $\mathcal{S}^\Delta := \mathcal{S} \cup \mathcal{S}^r$ and take $\Delta$ nearly as in the proof above. We only swap the left-hand sides and right-hand sides of $\Leftrightarrow$. The resulting set does not exclusively consist of $\mathcal{S}$-definitions, because some right-hand sides are not classical $\mathcal{S}$-formulae. But they do not define special predicates. Thus we obtain an $\mathcal{S}$-definition for every predicate symbol in $\mathcal{S}^r \backslash \mathcal{S}$ in the empty set of sentences. For every $\mathcal{S}$-structure $\mathcal{I}$, we have $\mathcal{I}^\Delta\lceil_{\mathcal{S}^r} = \mathcal{I}^r$, hence by Theor. 2.29 the assertion. □

# Chapter 3

# An Object-Oriented Data Model

The task of a data model is to give a means to describe data and operations defined on that data. So when we speak of data in this context, what do we really mean? Commonly in database management systems two fundamentally different types of information are present. First of all we store data in the traditional sense like pieces of goods or their part lists. Other types of information are used to give the former structure. It is *meta data* like the knowledge that every product has a price going along with it. The meta data is imperative when dealing with bulk data. The reasons are manifold and stretch over all requirements on database management systems mentioned in the introduction. Without structuring data, it is impossible to define index structures for the efficient evaluation of queries. Efficiently storing data without knowing the underlying structure is simply not feasible. For these reasons we divide the data into two parts. Meta data is data that is relatively stable over the time and describes the time-varying data. Thus giving formats for enumerations, rules to derive knowledge and semantic constraints to restrict the time-varying data and capture dynamic aspects.

## 3.1 Schema

Our data model describes two parts, a *schema* and an *instance*. The schema is meant to describe the part of the application domain that is relatively stable over a long period of time. It gives the structure for the application domain. The schema comprises basically only formulae, which we group according to their nature.

Classes are a means to structure the application domain. Since that structure is normally fixed over a long period, the subclass hierarchy and the signatures of the classes are part of the schema. The signatures of the classes constitute the interface of a class, defining the structural part of the objects of that class. Furthermore, the restrictions that are imposed on the time-varying parts are an ingredient of a schema. These restrictions can be divided into a part that is vital for technical reasons and a part that is relevant for the application. In this section we deal only with the former. The latter is discussed in Chap. 4.

This view of a schema is very similar to Liu's [Liu96]. But his definition lacks the possibility to declare semantic constraints, which are important for our work. It is also in accordance with many other object-oriented data models [Ban88, AK89, ABD$^+$89, Bee89, KL89b, CM90, GPvG90, ABGO93, LX93, ST93]. However, having based the data model on F-logic, we can exploit the flexibility, expressiveness, precise syntax and semantics, and proof theory to tailor our data model to meet our needs. These needs are among others the potency of conceiving formulae that talk about schema aspects.

Definition 3.1 (Schema)
*Let $\mathcal{L}$ be an F-logic language. A* (database) schema *$D$ for $\mathcal{L}$ is of the form*

$$D = \langle \mathrm{CLASS}_D | \mathrm{METH}_D | \mathrm{HIER}_D | \mathrm{SIG}_D | \mathrm{SC}_D \rangle$$

*where* $\mathrm{CLASS}_D, \mathrm{METH}_D, \mathrm{HIER}_D, \mathrm{SIG}_D, \mathrm{SC}_D \notin \mathcal{V} \cup \mathcal{S}$ *with*

- *a finite, non-empty set of constants,* $\mathrm{CLASS}_D \subset \mathcal{F}_0$*, for class names,*
- *a finite set of constants,* $\mathrm{METH}_D \subset \mathcal{F}_0$*, for method names,*
- *a finite set of ground is-a assertions,*

  $$\mathrm{HIER}_D \subset \{a :: b \mid a, b \in \mathrm{CLASS}_D\} \ ,$$

  *to form the class hierarchy,*
- *a finite set of signatures,*

  $$\mathrm{SIG}_D \subset \{c[m \mathbin{@} a_1, \ldots, a_k \approx\!\!> (r_1, \ldots, r_l)] \mid c, a_1, \ldots, a_k, r_1, \ldots, r_l \in \mathrm{CLASS}_D, \\ m \in \mathrm{METH}_D, k \geq 0, l \geq 1\} \ ,$$

  *to declare signatures for classes and methods, and*
- *a set of F-sentences,* $\mathrm{SC}_D$*, to restrict the set of allowed instances, which comprises besides application-dependent constraints as introduced in Chap. 4 the following:*
  - *the set of* unique name axioms

    $$\mathrm{UNA}_D = \{\overset{\circ}{\neq} (t, t') \mid t, t' \in U(\mathcal{F}), t \not\equiv t'\} \ ,$$

  - *the set of* definedness axioms

    $$\begin{aligned} \mathrm{DEF}_D = \{&\forall C\, \forall M \\ &\forall O\, \exists V \\ &(O[M \sim\!\!> V] \longleftarrow C[M \approx\!\!> (\,)] \wedge O : C)^{1} \\ &| \approx\!\!> \equiv \Rightarrow, \sim\!\!> \equiv \rightarrow \textit{ or } \approx\!\!> \equiv \Rrightarrow, \sim\!\!> \equiv \twoheadrightarrow\} \ , \end{aligned}$$

[1]By a misuse of notation variable $V$ may be $\{\,\}$ if $\sim\!\!>$ is $\twoheadrightarrow$.

– *and the set of* well-typedness axioms

$$
\begin{array}{rl}
\mathrm{WT}_D = & \{\forall O\,\forall M\,\forall A_1 \cdots \forall A_k\,\forall V \\
& \;\exists C\,\exists B_1 \cdots \exists B_k \\
& \;(O : C \wedge A_1 : B_1 \wedge \ldots \wedge A_k : B_k \wedge C[M\ @\,B_1, \ldots, B_k \mathrel{\approx\!>} (\,)] \\
& \;\longleftarrow \\
& \;O[M\ @\,A_1, \ldots, A_k \mathrel{\sim\!>} V])^2 \\
& \;\mid k \in \mathbb{N} \quad \textit{and} \quad \mathrel{\approx\!>} \equiv \Rightarrow, \mathrel{\sim\!>} \equiv \rightarrow \textit{ or } \mathrel{\approx\!>} \equiv \Rightarrow\!\!\!>, \mathrel{\sim\!>} \equiv \twoheadrightarrow\} \\
& \cup \\
& \{\forall O\,\forall M\,\forall A_1 \cdots \forall A_k\,\forall V \\
& \;\forall C\,\forall B_1 \cdots \forall B_k\,\forall W \\
& \;(V : W \\
& \;\longleftarrow \\
& \;C[M\ @\,B_1, \ldots, B_k \mathrel{\approx\!>} W] \wedge O[M\ @\,A_1, \ldots, A_k \mathrel{\sim\!>} V] \wedge \\
& \;O : C \wedge A_1 : B_1 \wedge \ldots \wedge A_k : B_k) \\
& \;\mid k \in \mathbb{N} \quad \textit{and} \quad \mathrel{\approx\!>} \equiv \Rightarrow, \mathrel{\sim\!>} \equiv \rightarrow \textit{ or } \mathrel{\approx\!>} \equiv \Rightarrow\!\!\!>, \mathrel{\sim\!>} \equiv \twoheadrightarrow\}\ .
\end{array}
$$

*The sets* $\mathrm{CLASS}_D$ *and* $\mathrm{METH}_D$ *are disjoint,* $\mathrm{CLASS}_D \cap \mathrm{METH}_D = \emptyset$.

When the language and the schema are understood from the context, we will not mention them explicitly in subsequent sections.

The set $\mathrm{CLASS}_D$ serves to declare the class names used in the application domain. The classes form a hierarchy, which is defined by the set $\mathrm{HIER}_D$. The structural part of classes is defined by signatures in the set $\mathrm{SIG}_D$. These signatures are used for a special kind of semantic constraints, $\mathrm{WT}_D$, that enforces typing in method invocations. Here we do not follow the approach laid out in Sect. 2.4 of using special models, rather we use the semantic constraints in $\mathrm{WT}_D$, which connect method invocations and method signatures. This relationship will be discussed later on in more detail. Due to the F-logic semantics of ::, subclasses inherit all signatures from their superclasses, and each element of a subclass is also an element of the corresponding superclasses.

Semantic constraints defined in the set $\mathrm{SC}_D$ identify "meaningful schemas" and restrict the set of instances for the schemas. Part of the semantic constraints are unique name axioms [Rei80] for the elements of the Herbrand universe. These ensure that different ground id-terms stand for different objects. The reason for this is that we want to derive conclusions based on syntactic material, namely Herbrand bases, assuming that syntactically different things are also semantically different. The existence of unique name axioms has an effect on the equality theory. The equality predicate $\overset{\circ}{=}$ is even diagonal on the constants of the Herbrand universe. This entails that a schema has an acyclic class hierarchy if we demand that the set $\mathrm{HIER}_D$ is a model of the unique name axioms, $\mathrm{HIER}_D \models \mathrm{UNA}_D$, which we do throughout this work.

There are two different kinds of well-typedness axioms. The first kind is a formalisation of the notion that a data atom is covered by a signature expression. This has

[2]Again by a misuse of notation variable $V$ may be $\{\,\}$ if $\mathrel{\sim\!>}$ is $\twoheadrightarrow$.

to hold for every data atom in a model of $\mathrm{WT}_D$. Therefore the first condition 2a in Def. 2.24 is satisfied. The second kind is a formalisation of condition 2b in Def. 2.24. Theorem 3.9 shows the correspondence in more detail.

The set of definedness axioms is introduced to prevent the occurrence of null values, i. e. if an object is element of some class, the method invocation for a method that is declared for the class and takes no arguments has to return a value.

The set of semantic constraints is infinite for two reasons. The set of unique name axioms is infinite if the Herbrand universe $U(\mathcal{F})$ is infinite. The set $\mathrm{WT}_D$ is infinite by definition. As we will see later in Sect. 3.2 instances of a schema may be infinite. But this means that we cannot give an a priori upper bound for the highest possible arity of methods.

The sets $\mathrm{UNA}_D$, $\mathrm{WT}_D$ and $\mathrm{DEF}_D$ are independent of the underlying schema $D$. The sets $\mathrm{WT}_D$ and $\mathrm{DEF}_D$ are even independent of the underlying language. So we define the set of axioms

$$\mathrm{AX} := \mathrm{UNA}_D \cup \mathrm{WT}_D \cup \mathrm{DEF}_D$$

independently of the underlying schema.

We mention above that a schema includes rules to derive further knowledge. These rules lack in our definition of a schema. As our focus is on different aspects of database design, we will not pursue this issue in this work. It is possible to derive a view mechanism based on Def. 3.13.

It is possible to minimise the sets $\mathrm{HIER}_D$ and $\mathrm{SIG}_D$, by removing redundancies. For instance, the set $\{a :: b, b :: c, a :: c\}$ can be reduced to the set $\{a :: b, b :: c\}$ because the is-a assertion $a :: c$ is a logical consequence of the two is-a assertions $a :: b$, $b :: c$. Similarly, it is possible to reduce the size of the set $\mathrm{SIG}_D$.

Example 3.2

*The entity-relationship schema in Exam. 1.2 can be modelled as schema with the following components:*

$$\begin{aligned}
\mathrm{CLASS} &:= \{\mathsf{Person}, \mathsf{Phone\text{-}admin}, \mathsf{Faculty}, \mathsf{School}, \mathsf{Phone}, \mathsf{Department}\},\\
\mathrm{METH} &:= \{\mathsf{children}, \mathsf{fac}, \mathsf{sch}, \mathsf{ph}, \mathsf{dep}\},\\
\mathrm{HIER} &:= \{\mathsf{Faculty} :: \mathsf{Person}\},\\
\mathrm{SIG} &:= \{\mathsf{Person}[\mathsf{children} \mathbin{@} \mathsf{Person} \Rightarrow\!\!\!\Rightarrow \mathsf{Person}],\\
&\qquad \mathsf{Phone\text{-}admin}[\mathsf{fac} \Rightarrow \mathsf{Faculty}; \mathsf{sch} \Rightarrow \mathsf{School};\\
&\qquad\qquad \mathsf{ph} \Rightarrow \mathsf{Phone}; \mathsf{dep} \Rightarrow \mathsf{Department}]\}\ .
\end{aligned}$$

*We added to the schema the class* Person *as a superclass of the class* Faculty*, although this is-a relationship is not present in Fig. 1.5. The class* Person *has one set-valued method, which takes as argument an object of class* Person *and returns a set of* Person*s. This set represents the children of the* Person *host object.*

*Its "schema graph" is depicted in Fig. 3.1. Here arrows like* $\dashrightarrow$ *denote that the class at the bottom of the arrow is a subclass of the class the arrow is pointing to. The arrows like* $\xrightarrow{\mathsf{m}}$ *denote signature expressions for scalar methods that take no arguments. The class at the butt is the class the method m is declared for, and the class at the point is a result class for that method.*

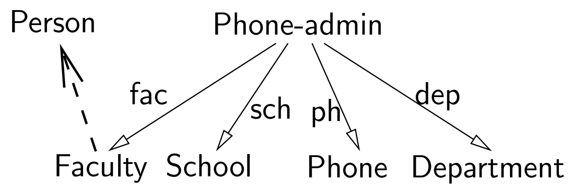


Figure 3.1: The schema graph of the F-logic schema

*We used the first step of the method presented in [BMP96] to generate this schema. The basic idea underlying this step is that entity sets are mapped onto classes and relationship sets are mapped onto classes with appropriate method declarations. Additionally some semantic constraints are needed for the classes introduced for relationship sets. As we present these constraints first in Chap. 4, we leave them out here.*

In principle a schema $D$ is a set of F-formulae. These formulae are classified according to their nature. For convenience, we sometimes write $D \cup \Gamma$ for a schema $D$ and a set of F-formulae $\Gamma$ to denote the schema resulting when we include the formulae in $\Gamma$ into $D$. The nature of the set of formulae $\Gamma$ is taken from the context in case of ambiguities.

Scalar methods that take no arguments play an important rôle in this work. We shall call these scalar methods *attributes.* Sometimes it is important to guarantee that an attribute is only defined for exactly one class. These attributes are *proper* attributes.

Definition 3.3 (Attribute)
*Let $D$ be a schema.*

- *A method $m \in \mathrm{METH}_D$ is called an* attribute *for class $c \in \mathrm{CLASS}_D$ :iff the following holds:*
$$\mathrm{HIER}_D \cup \mathrm{SIG}_D \models c[m \Rightarrow ()] \ .$$
- *A method $m \in \mathrm{METH}_D$ is called a* proper attribute *for class $c \in \mathrm{CLASS}_D$ :iff all of the following holds:*
  - $\mathrm{SIG}_D \models c[m \Rightarrow ()]$, *and*
  - $\mathrm{SIG}_D \not\models d[m \Rightarrow ()]$, *if $c \not\equiv d$.*

*The set of attributes $\{m \in \mathrm{METH}_D \mid \mathrm{HIER}_D \cup \mathrm{SIG}_D \models c[m \Rightarrow ()]\}$ for a class $c$ is denoted $\mathrm{Attr}_D(c)$. The set of all attributes $\bigcup_{c \in \mathrm{CLASS}_D} \mathrm{Attr}_D(c)$ is denoted by $\mathrm{Attr}_D$.*

Note that the polymorphism of F-logic allows a method $m \in \mathrm{METH}_D$ to occur as attribute and as method that takes arguments.

In the preceding definition we used the logical entailment to make assertions about properties of elements of a schema. Because of our restrictions on the material used in a schema, it is possible to decide the implication using only simplified resolution rules. For these the implication problem is decidable.

Theorem 3.4
*Let $D$ be a schema. The implications $\mathrm{HIER}_D \models c::d$ and $\mathrm{HIER}_D \cup \mathrm{SIG}_D \models c[m \Rightarrow d]$ are decidable.*

PROOF. To prove the hypothesis, we can devise an algorithm based on the inference rules given for F-logic. Due to the finiteness of the input $\mathrm{HIER}_D$ and $\mathrm{SIG}_D$ and the form of the applicable inference rules (IS-A reflexivity, IS-A acyclivity, IS-A transitivity, type inheritance, input restriction and output relaxation), we can exhaustively apply the inference rules on the input and stop after a priori finitely many steps. If the molecule to be tested is an element of the result, the implications hold otherwise the implications do not hold.

□

## 3.2 Instance

Having described the part of our data model that is relatively stable over a long period of time, we come now to the time-varying part, the *instance*, that comprises objects and their method values. An instance consists of two different kinds of F-formulae. The first kind populates classes and the second kind defines method values for these objects. Furthermore, an instance has to obey the semantic constraints set out by its schema. To define an instance, we start off by defining the syntactic part of an instance, the *extension.*

DEFINITION 3.5 (EXTENSION)
*Let $D$ be a schema for a language $\mathcal{L}$. An* extension $f \in \mathrm{ext}(D)$ *of the schema $D$ is of the form*

$$f = \langle \mathrm{pop}_f | \mathrm{ob}_f \rangle$$

*where* $\mathrm{pop}_f, \mathrm{ob}_f \notin \mathcal{V} \cup \mathcal{S}$ *with*

- *a set of ground is-a assertions,*

$$\mathrm{pop}_f \subset \{o : c \mid o \in \mathcal{F}_0 \backslash (\mathrm{CLASS}_D \cup \mathrm{METH}_D), c \in \mathrm{CLASS}_D\} \ ,$$

  *populating classes, and*

- *a set of ground object molecules with data expressions,*

$$\begin{aligned} \mathrm{ob}_f \subset \{o[m \ @\, a_1, \ldots, a_k \leadsto r] \mid {} & k \geq 0 \quad \textit{and} \quad m \in \mathrm{METH}_D \quad \textit{and} \\ & o, a_1, \ldots, a_k, r \in \mathcal{F}_0 \backslash (\mathrm{CLASS}_D \cup \mathrm{METH}_D) \ \textit{and} \\ & \mathrm{pop}_f \models \exists C\, \exists C_1 \cdots \exists C_k\, \exists R \\ & \qquad (o{:}C \wedge a_1{:}C_1 \wedge \cdots \wedge a_k{:}C_k \wedge r{:}R)\} \\ \cup \quad & \\ \{o[m \ @\, a_1, \ldots, a_k \twoheadrightarrow \{\,\}] \mid {} & k \geq 0 \quad \textit{and} \quad m \in \mathrm{METH}_D \quad \textit{and} \\ & o, a_1, \ldots, a_k \in \mathcal{F}_0 \backslash (\mathrm{CLASS}_D \cup \mathrm{METH}_D) \ \textit{and} \\ & \mathrm{pop}_f \models \exists C\, \exists C_1 \cdots \exists C_k \\ & \qquad (o{:}C \wedge a_1{:}C_1 \wedge \cdots \wedge a_k{:}C_k)\} \ , \end{aligned}$$

  *for the definition of method values for objects.*

The extension level of a database is restricted to facts for populating elements of $\mathrm{CLASS}_D$ and for defining the values for method invocations. We do not permit the declaration of new classes, changes to the class hierarchy, or the introduction of new signatures. We bound id-terms occurring in an extension to constants. This limitation conforms to the definition of a schema, where it is not possible to deduct knowledge from the internal structure of id-terms.

Alas, objects and classes have the same domain and are interwoven together such that there is no clear separation of the notions of schema and extension in F-logic. Therefore, we introduce this separation in our data model.

Next we define what can be deduced from an extension and its schema.

Definition 3.6 (Completion)
*Let $D$ be a schema and $f$ be an extension of the schema $D$. The* completion of $f$ under $D$ *is*

$$\begin{aligned}\mathrm{compl}_D(f) := \{\alpha \in \mathcal{HB}(\mathcal{F}) \mid\ & \mathrm{CLASS}_D \cup \mathrm{METH}_D \cup \mathrm{HIER}_D \cup \mathrm{SIG}_D \cup \\ & \mathrm{pop}_f \cup \mathrm{ob}_f \models \alpha\}\ .\end{aligned}$$

The definition above is somewhat sloppy since class names, the elements in $\mathrm{CLASS}_D$, and method names, the elements in $\mathrm{METH}_D$, are not F-formulae. In this context we see every class name $c \in \mathrm{CLASS}_D$ and method name $m \in \mathrm{METH}_D$ as object molecule with no method expression, $c[\,]$ and $m[\,]$, respectively. The addition of all class names and method names for the evaluation of the completion of an extension ensures that every class name and every method name is activated in its completion.

Theorem 3.7
*Let $f$ be an extension of a schema $D$. The completion of $f$ under $D$ is the smallest H-model of* $\mathrm{CLASS}_D \cup \mathrm{METH}_D \cup \mathrm{HIER}_D \cup \mathrm{SIG}_D \cup \mathrm{pop}_f \cup \mathrm{ob}_f$ *with respect to the set inclusion $\subset$.*

Proof. It is clear that the set $\mathrm{compl}_D(f)$ is an H-structure. The set $\mathrm{compl}_D(f)$ is a subset of the Herbrand base, $\mathrm{compl}_D(f) \subset \mathcal{HB}(\mathcal{F})$, by definition and also closed under the logical implication by definition. It is also trivial that $\mathrm{compl}_D(f)$ is an H-model of $\mathrm{CLASS}_D \cup \mathrm{METH}_D \cup \mathrm{HIER}_D \cup \mathrm{SIG}_D \cup \mathrm{pop}_f \cup \mathrm{ob}_f$.

Whenever there is an H-model $\mathbf{H}$ of $\mathrm{CLASS}_D \cup \mathrm{METH}_D \cup \mathrm{HIER}_D \cup \mathrm{SIG}_D \cup \mathrm{pop}_f \cup \mathrm{ob}_f$, every ground molecule $\alpha$ implied by $\mathrm{CLASS}_D \cup \mathrm{METH}_D \cup \mathrm{HIER}_D \cup \mathrm{SIG}_D \cup \mathrm{pop}_f \cup \mathrm{ob}_f$ is element of $\mathbf{H}$, $\alpha \in \mathbf{H}$, by definition, and hence $\mathrm{compl}_D(f) \subset \mathbf{H}$, since $\mathrm{compl}_D(f)$ comprises all implied ground molecules of $\mathrm{CLASS}_D \cup \mathrm{METH}_D \cup \mathrm{HIER}_D \cup \mathrm{SIG}_D \cup \mathrm{pop}_f \cup \mathrm{ob}_f$ by definition.

$\square$

An instance of a schema is an extension that obeys the semantic constraints declared in that schema.

Definition 3.8 (Instance)
*Let $f$ be an extension of a schema $D$. Then $f$ is called a(n)* (allowed) (database) instance *of $D$, $f \in \mathrm{sat}(D)$, if* $\mathrm{compl}_D(f)$ *is an H-model of* $\mathrm{SC}_D$.

The definition of method values in instances is always materialised. We do not allow in this definition the evaluation of method values. But this is not a problem for us, because we are especially interested in directly defined method values.

In Sect. 2.4 dealing with typing aspects in F-logic, we presented well-typed programmes. The link between well-typed programmes and instances are the completions of the latter under the schemas of the latter.

THEOREM 3.9
*Let $f$ be an instance of a schema $D$. The completion* $\mathrm{compl}_D(f)$ *is a typed H-structure.*

PROOF. The completion $\mathrm{compl}_D(f)$ is an H-model of the semantic constraints $\mathrm{SC}_D$. These semantic constraints include the well-typedness axioms, which ensure that first every data atom in $\mathrm{compl}_D(f)$ is covered by a signature and second that if a data atom is covered by a signature the result value is of the required type. $\square$

The set $\mathrm{CLASS}_D \cup \mathrm{METH}_D \cup \mathrm{HIER}_D \cup \mathrm{SIG}_D \cup \mathrm{pop}_f \cup \mathrm{ob}_f$ is simply a set of F-formulae, so an F-programme. The theorem above can now be used to make statements about $\mathrm{CLASS}_D \cup \mathrm{METH}_D \cup \mathrm{HIER}_D \cup \mathrm{SIG}_D \cup \mathrm{pop}_f \cup \mathrm{ob}_f$ as well-typed programmes. A prerequisite for that is an appropriate definition of canonic models. In this work we define the completion of the instance $f$ under the schema $D$ to be the canonical model of the set $\mathrm{CLASS}_D \cup \mathrm{METH}_D \cup \mathrm{HIER}_D \cup \mathrm{SIG}_D \cup \mathrm{pop}_f \cup \mathrm{ob}_f$, thus the completion $\mathrm{compl}_D(f)$ is the only typed-canonic model and therefore $\mathrm{CLASS}_D \cup \mathrm{METH}_D \cup \mathrm{HIER}_D \cup \mathrm{SIG}_D \cup \mathrm{pop}_f \cup \mathrm{ob}_f$ is a well-typed programme.

EXAMPLE 3.10
*An instance of the schema in Exam. 3.2 can be composed of the following components:*

$$\begin{aligned} \mathrm{pop}_f &:= \{\mathsf{bob} : \mathsf{Faculty}, \mathsf{alice} : \mathsf{Person}, \mathsf{john} : \mathsf{Person}\}, \\ \mathrm{ob}_f &:= \{\mathsf{bob}[\mathsf{children} \mathbin{@} \mathsf{alice} \twoheadrightarrow \mathsf{john}], \\ &\qquad \mathsf{alice}[\mathsf{children} \mathbin{@} \mathsf{bob} \twoheadrightarrow \mathsf{john}]\} \ . \end{aligned}$$

The set of all objects for an extension and the set of classes an object is element of in an extension is defined below.

DEFINITION 3.11
*Let $f$ be an extension of a schema $D$.*

- *The set of all objects for extension $f$ is denoted* $\mathrm{obj}(f) := \{o \mid \textit{ex. } c\text{: } o : c \in \mathrm{pop}_f\}$.
- *The* class-label $\mathrm{l_{Cl}}(o)$ *for an object $o \in \mathrm{obj}(f)$ is the set of classes object $o$ is element of,* $\mathrm{l_{Cl}}(o) := \{c \mid \mathrm{HIER}_D \cup \mathrm{pop}_f \models o : c\}$.

The next lemma justifies the use of only some syntactic material in the deduction of knowledge from schemas and extensions.

LEMMA 3.12
*Let $f$ be an extension of a schema $D$. Then $o \in \mathrm{obj}(f)$ and $c \in \mathrm{l_{Cl}}(o)$ iff* $\mathrm{compl}_D(f) \models$ *$o$:$c$.*

PROOF. The assertion follows from the proof theory of F-logic.
$\square$

## 3.3 Queries and Views

In this section we present the first and only operation that is defined for our data model. It is a fundamental must for databases because it allows us to retrieve information stored in a database instance of a schema.

Definition 3.13 (Query)
*Let $D$ be a schema. A set*

$$Q \subset \{H \longleftarrow T \mid H \longleftarrow T \textit{ is an F-Horn-rule}\}$$

*is a* query over $D$*.*

The definition of queries is based on arbitrary F-Horn-rules. This allows to extract information about instances as well as schemas. This is due to the great flexibility that F-logic offers. It is credited to the uniform framework in which F-logic deals with signatures and data. Though some condemn this flexibility (cf. [AK92]) because it seems to come with the trade off that no strict typing is possible in earlier versions of F-logic [Mai86, KW89, KL89a]. In fact, that is not the case in the current version of F-logic as presented in [KLW95].

In general we could define views based on queries, following the ideas in the work of dos Santos [dSAD94] on views in object-oriented data models; but we refrain from doing so, and in addition we even present only a simplified version of a semantics for queries, which is based on the immediate consequence operator for non-recursive F-Horn-rules. The rationale for this simplified semantics lies in that we use queries in Chap. 7 to construct new instances. However, syntactically, the F-Horn-rules in those queries are recursive, although their semantical nature is intended to be non-recursive.

Example 3.14
*The set $Q := \{O{:}d \longleftarrow O{:}d\}$ is a query in the sense above. We use this query later in Chap. 7 to populate the class d in a new instance based on the population of the class d in an original instance. So in fact, although the syntactic form of the rule is recursive, we have to read the rule for this application as $O{:}d_{new} \longleftarrow O{:}d_{old}$, which is non-recursive.*

The following is a simple but for our needs sufficient approach to define the semantics of queries.

Definition 3.15 (Eval)
*Let $f = \langle \mathrm{pop}_f | \mathrm{ob}_f \rangle$ be an instance of a schema $D$, and $Q$ be a query over $D$. We define*

$$\mathrm{eval}(Q, f) := \bigcup_{H \longleftarrow T \in Q} \{\nu(H) \mid \mathrm{compl}_D(f) \models \nu(T)\} \ .$$

Since $\mathrm{compl}_D(f)$ is an H-structure and satisfies the unique name axioms, its universe is a subset of the Herbrand universe $U(\mathcal{F})$, and so $\nu(H)$ and $\nu(T)$ are a well-formed ground molecules.

Because we do not need concepts like updates or transactions in this work, we do not define these concepts.

Sometimes we have two schemas the structures of which are nearly identical. Then *projection queries* help to populate the extensions of one schema by means of the extensions of the other schema.

Definition 3.16 (Projection Query)
*Let $C, A \subset \mathcal{F}_0$ be two sets of constants. The query*

$$\mathrm{proj}_{C,A} := \{O{:}c \longleftarrow O{:}c \mid c \in C\} \cup \{O[m \to R] \longleftarrow O[m \to R] \mid m \in A\}$$

*is called* projection query.

*If $C = \mathrm{CLASS}_D$ and $A = \mathrm{METH}_D$ for a schema $D$, then $\mathrm{id}_D := \mathrm{proj}_{C,A}$ is called* identity query.

The projection of an extension is an extension as is shown in the following lemma.

Lemma 3.17
*Let $f$ be an extension of a schema $D$, $C \subset \mathrm{CLASS}_D$ be a set of classes, and $A \subset \mathrm{METH}_D$ be a set of attributes.*

1. *Then for all classes $c \in C$*

$$o{:}c \in \mathrm{compl}_D(f) \text{ iff } o{:}c \in \mathrm{eval}(\mathrm{proj}_{C,A}, f)$$

*and for all attributes $m \in A$*

$$o[m \to r] \in \mathrm{compl}_D(f) \text{ iff } o[m \to r] \in \mathrm{eval}(\mathrm{proj}_{C,A}, f)$$

*hold.*

2. *Then the evaluation $\mathrm{eval}(\mathrm{proj}_{C,A}, f)$ is an extension of the schema $D$.*

Proof for 1.

$$\begin{aligned} o{:}c \in \mathrm{compl}_D(f) &\text{ iff } \mathrm{compl}_D(f) \models o{:}c \quad \text{(Def. 2.23)} \\ &\text{ iff ex. } \nu \text{ with } \nu(O) \equiv o{:} \\ &\qquad \mathrm{compl}_D(f) \models \nu(O{:}c) \\ &\text{ iff } o{:}c \in \mathrm{eval}(\mathrm{proj}_{C,A}, f) \quad (O{:}c \longleftarrow O{:}c \in \mathrm{proj}_{C,A} \text{ and Def. 3.15}) \end{aligned}$$

$$\begin{aligned} o[m \to r] \in \mathrm{compl}_D(f) &\text{ iff } \mathrm{compl}_D(f) \models o[m \to r] \quad \text{(Def. 2.23)} \\ &\text{ iff ex. } \nu \text{ with } \nu(O) \equiv o \text{ and } \nu(R) \equiv r{:} \\ &\qquad \mathrm{compl}_D(f) \models \nu(O[m \to R]) \\ &\text{ iff } o[m \to r] \in \mathrm{eval}(\mathrm{proj}_{C,A}, f) \\ &\qquad (O[m \to R] \longleftarrow O[m \to R] \in \mathrm{proj}_{C,A} \text{ and Def. 3.15}) \end{aligned}$$

Proof for 2. If $o{:}c, o[m \to r] \in \mathrm{eval}(\mathrm{proj}_{C,A}, f)$, then, due to 1, $o{:}c, o[m \to r] \in \mathrm{compl}_D(f)$. Therefore the molecular formulae $o{:}c$ and $o[m \to r]$ comply with the restrictions imposed on elements of extensions of the schema $D$.

$\square$

We close this section with the observation that whenever the completions of two instances are the same, their query results will be equal.

Lemma 3.18 (Eval Function)
*Let $f$ and $f'$ be two extensions of a schema $D$. Whenever the completions of the instances $f$ and $f'$ are equal,* $\mathrm{compl}_D(f) = \mathrm{compl}_D(f')$*, then the query evaluations under the instances are equal,* $\mathrm{eval}(Q, f) = \mathrm{eval}(Q, f')$*.*

Proof. This lemma follows from the definition of a evaluation $\mathrm{eval}(Q, \cdot)$.

□

# Chapter 4

# Application-Dependent Constraints

As mentioned previously, the semantic constraints of a schema can be divided into two parts. Both parts are necessary to identify meaningful schemas and to restrict the set of instances of schemas such that restrictions of the application domain are reflected. In the relational world there are a number of well understood classes of semantic constraints (functional dependencies, multi-valued dependencies, join dependencies, inclusion dependencies). They do not miss in any standard database textbook. These constraints have been profoundly investigated. The theory of database design has benefited from these investigations. A number of normal forms based on semantic constraints characterise desirable relational database schemas. Alas, the theory of database design for object-oriented databases lacks such clear and well-understood normal forms due to the lack of well-understood classes of semantic constraints. We surmise that the origins of object-oriented databases from object-oriented programming led to a plethora of database operations. This newly found freedom in expressing restrictions seduced to burden nearly all semantic constraints on the shoulders of application programmers, which hid these in the program logic of the applications.

In this work we want to use *path functional dependencies*, a class of semantic constraints, which can be understood as extension of functional dependencies for object-oriented database models. Path functional dependencies were introduced by Weddell [Wed89, Wed92].

We employ path functional dependencies for the design of object-oriented database schemas where they occur among others in the output of a schema transformation called *pivoting* defined in Sec. 7.1. This transformation is meant to improve object-oriented schemas. To obtain path functional dependencies in the output of pivoting, the input schema must contain path functional dependencies. A second kind of semantic constraints generated by the transformation pivoting are *onto constraints*. Their nature is completely different from path functional dependencies. They are more like inclusion dependencies as known from the relational world. The concept of an onto constraint has been mentioned by Thalheim [Tha93a], but without naming it.

Although we motivate the existence of onto constraints and path functional dependencies in their connection to pivoting, they can exist independent of this transformation. For path functional dependencies examples are given by Weddell; they may serve various purposes like for example query optimisation.

In a given schema, we explicitly define semantic constraints. Beyond that there are additional constraints that hold for every instance. Therefore we say a set of semantic constraints $\Gamma$ implies a constraint $\gamma$ under a schema $D$ if and only if all instances of the schema $D$ satisfying the set of constraints $\Gamma$ satisfy the constraint $\gamma$. This notion of implication is very similar to the logical implication denoted $\models$. Unfortunately, the correspondence is not as close as in the relational data model. In our framework only the following holds.

Lemma 4.1
*Let $D$ be a schema with minimal semantic constraints, i. e. $\mathrm{SC}_D = \mathrm{AX}$, and $\mathrm{SC}$ and $\mathrm{SC}'$ be two sets of sentences. Then $\mathrm{sat}(D \cup \mathrm{SC}) \subset \mathrm{sat}(D \cup \mathrm{SC}')$ if $\mathrm{SC} \models \mathrm{SC}'$.*

Proof.

$$\begin{array}{lll} f \in \mathrm{sat}(D \cup \mathrm{SC}) & \text{iff} & \mathrm{compl}_D(f) \models \mathrm{AX} \cup \mathrm{SC} \\ & \text{then} & \mathrm{compl}_D(f) \models \mathrm{AX} \cup \mathrm{SC}' \\ & \text{iff} & f \in \mathrm{sat}(D \cup \mathrm{SC}') \ . \end{array}$$

□

Unfortunately, the converse does not hold as demonstrated in the next example.

Example 4.2
*Let $D$ be a schema with $\mathrm{HIER}_D := \{a{::}b\}$, and two sets of semantic constraints $\mathrm{SC} := \{b{::}c\}$ and $\mathrm{SC}' := \{b{::}d\}$. Additionally, let $f$ be an extension of the schema $D$. When we look at the completion of the extension $f$ under the schema $D$, we observe the only non-trivial class is-a assertion found in the completion is the molecule $a{::}b$, since there is only one in the syntactic material of the schema $D$, namely the molecule $a{::}b$, and non-trivial class is-a assertions can only be implied from class is-a assertions. Consequently, it is clear that $\mathrm{sat}(D \cup \mathrm{SC}) = \mathrm{sat}(D \cup \mathrm{SC}') = \emptyset$ holds, but neither $\mathrm{SC} \models \mathrm{SC}'$ nor $\mathrm{SC}' \models \mathrm{SC}$ hold.*

Both sets of semantic constraints SC and SC′ include assertions over the schema $D$, but, unfortunately, the schema $D$ does not supply the schema element requested by the semantic constraints. This mismatch takes its root in that we can only define formulae over an alphabet not over the structural part of a schema. We might think we could exclude formulae from the semantic constraints based on our knowledge on the structural part of a schema, because we can easily spot the discrepancy between the is-a assertions in the class-hierarchy-giving part of the schema and the semantic constraints in Exam. 4.2. However, problems arise when formulae are non-ground unless we want to put serious restrictions onto the formulae employed as semantic constraints. In that case we need a schema-relativism of the formulae, which stretches over formulae not only domain elements.

In the relational data model it is indeed possible to define the notion of a formula over the structural part of a schema, because the structural part of the schema constitutes the underlying alphabet. Thus we can conclude from the relationship between sets of allowed instances what the relationship between the respective sets of semantic constraints in terms of the logical entailment is, i. e., the converse of Lem. 4.1 can be established in the relational data model. Consequently, we have to consider interpretations and formulae over all schemas. Despite this fact, we speak about formulae over schemas indicating that the underlying languages are identical.

Common to all schemas are unique name axioms, well-typedness axioms and definedness axioms. The well-typedness axioms and definedness axioms do not even depend on the language used.

## 4.1 Onto Constraints

For pivoting, the decomposing transformation studied in Sect. 7.1, we need so-called *onto constraints* on grounds of technical requirements, which will become clear when pivoting is described formally. An onto constraint guarantees that all objects of a result class of an attribute are referred to by an object of the class the attribute is defined for. Thus onto constraints are a special type of inclusion dependencies. They correspond to unary inclusion dependencies, which connect only singleton sets of attributes.

EXAMPLE 4.3
*We come back to Exam. 3.2 and extend the presented schema by semantic constraints that reflect restrictions imposed on its instances. An example for an onto constraint is*

$$\mathsf{Phone\text{-}admin}\{\mathsf{fac}|\mathsf{Faculty}\} \ .$$

*It states that every* Faculty *must have a phone.*[1]

DEFINITION 4.4 (ONTO CONSTRAINT)
*Let $D$ be a database schema, and $c \in \mathrm{CLASS}_D$ be a class.*

**Syntax:** *An* onto constraint *for $c$ over $D$ is of the form $c\{m|c_m\}$, where $m \in \mathrm{Attr}_D(c)$ is an attribute for $c$ and $c_m \in \mathrm{CLASS}_D$ is a class.*

**Semantics:** *The* onto constraint F-formula *for an onto constraint $c\{m|c_m\}$ is*

$$\forall V \exists O (O{:}c[m \to V] \land c[m \Rightarrow ()] \longleftarrow V{:}c_m) \ .$$

*The set of onto constraint F-formulae for $c$ in $\mathrm{SC}_D$ is denoted $\mathrm{OC}_D(c)$. $\mathrm{OC}_D$ denotes the set of all onto constraint F-formulae in $\mathrm{SC}_D$.*

Often we blur the clear distinction between syntax and semantics and write $c\{m|c_m\} \in \mathrm{OC}_D(c)$ whenever that is appropriate.

In the definition above there is no connection between the class $c_m$ and the rest of the definition, i. e., $c_m$ is not necessarily a result class of the attribute $m$ for the class $c$ although we might expect such a connection. When we look at the inference rules given in Def. 4.7 below, we see that inference rule "range restriction" produces onto

[1]Remember this is only hypothetical.

constraints with only a weak connection between the class $c_m$ on the one hand and the class $c$ and the attribute $m$ on the other hand.

We observe the similarity between definedness constraints and onto constraints when it comes to their semantics. The scope of these constraints stretches over two classes. The former demand that all attributes return a proper value, and the latter demand that an object is referred to by another object.

To obtain an algorithmic decision procedure for the implication of onto constraints, we have to conceive inference rules. However, the implication of onto constraints does not always yield onto constraints from a given set of onto constraints.

Example 4.5
*Let $D$ be a schema with onto constraint $d\{m|c\} \in \mathrm{OC}_D$ and signature atom $\mathrm{HIER}_D \cup \mathrm{SIG}_D \models d[m \Rightarrow c']$. Then this clearly implies the formula $\forall O(O : c' \longleftarrow O : c)$, because all elements of class $c$ have to be referred to by an object of class $d$, which entails that the referred object of class $c$ is also an object of class $c'$ due to the signature atom and the well-typedness axioms.*

We formalise formulae as presented in the preceding example as *class inclusion constraints.*

Definition 4.6 (Class Inclusion Constraint)
*Let $D$ be a schema, and $c, c' \in \mathrm{CLASS}_D$ be two classes.*

**Syntax:** *A* class inclusion constraint *(CIC) for $c, c'$ over $D$ is of the form $c \subset c'$.*

**Semantics:** *A* class inclusion constraint F-formula *for a class inclusion constraint $c \subset c'$ is*

$$\forall O(O : c' \longleftarrow O : c) \ .$$

*The symbol $\mathrm{CIC}_D$ denotes the set of all class inclusion constraint F-formulae in the set $\mathrm{SC}_D$ of semantic constraints.*

In this section we restrict the set of semantic constraints of a schema to class inclusion constraints and onto constraints plus unique name axioms, well-typedness axioms and definedness axioms until[2] otherwise mentioned, i. e.

$$\mathrm{SC}_D = \mathrm{AX} \cup \mathrm{CIC}_D \cup \mathrm{OC}_D \ .$$

Since the presence of the axioms has a more technical flavour, we shall often speak of semantic constraints meaning the constraints that are not axioms but application-dependent constraints.

Having class inclusion constraints, we can give inference rules for class inclusion constraints and onto constraints. The first rule incarnates the transitive nature of class inclusion constraints. The second rule captures the influence of the class hierarchy: if a class is a subclass of another class, each element of the subclass must be an element of its superclass. As the class hierarchy depends solely on the underlying schema, this rule

[2]In Sec. 4.4.

introduces a dependency of the inference mechanism on the underlying schema. The next rule embodies the interplay between onto constraints and signature declarations as shown in Exam. 4.5. Its effect is the introduction of class inclusion constraints orthogonal to the class hierarchy. While the last three rules saw class inclusion constraints as results, the next two rules deal with the inference of onto constraints. The first of these rules allows a restriction of the range of an onto constraint, whereas the second grants the relaxation of the domain of an onto constraint.

Given a set of class inclusion constraints and onto constraints over a schema the derivation from this set is not influenced by the set of constraints in the schema itself.

Definition 4.7 (Inference Rules for CICs and OCs)
*Let $\Upsilon \cup \{\upsilon\}$ be a set of class inclusion constraints and onto constraints over a schema $D$. The constraint $\upsilon$ is* derivable *from $\Upsilon$, written $\Upsilon \vdash_D \upsilon$,* iff *it is a member of $\Upsilon$ or is the result of one or more applications of the following inference rules.*

**C1.** C-transitivity: *For classes $c, c', c'' \in \mathrm{CLASS}_D$, if both $c \subset c'$ and $c' \subset c''$ can be derived, then so can $c \subset c''$.*

**C2.** Subclass inclusion: *For classes $c, c' \in \mathrm{CLASS}_D$, where $\mathrm{HIER}_D \models c :: c'$, derive $c \subset c'$.*

**C3.** Signature inclusion: *For classes $c, c' \in \mathrm{CLASS}_D$, if $d\{m|c\}$ can be derived, where $\mathrm{HIER}_D \cup \mathrm{SIG}_D \models d[m \Rightarrow c']$, then so can $c \subset c'$.*

**C4.** Range restriction: *For classes $d, c \in \mathrm{CLASS}_D$, if $c \subset c'$ and $d\{m|c'\}$ can be derived, then so can $d\{m|c\}$.*

**C5.** Domain relaxation: *For classes $d, c' \in \mathrm{CLASS}_D$, if $c \subset c'$ and $c\{m|d\}$ can be derived, where $m \in \mathrm{Attr}_D(c')$, then so can $c'\{m|d\}$.*

In this definition we bring the logical entailment to bear to extract knowledge about the class hierarchy and the signature of classes; but we have to keep in mind here that these extractions can be algorithmically decided because of Theor. 3.4.

An example for the application of the inference rules is shown in Exam. 4.10.

The *closure* of a set of class inclusion constraints and onto constraints contains all class inclusion constraints and onto constraints that can be derived from the set of class inclusion constraints and onto constraints.

Definition 4.8 (Closure)
*Let $\Upsilon$ be a set of class inclusion constraints and onto constraints over a schema $D$. The* closure *of $\Upsilon$ over the schema $D$, written $\Upsilon^{+_D}$, is the set of all class inclusion constraints and onto constraints $\upsilon$ over the schema $D$ where $\Upsilon \vdash_D \upsilon$.*

Although the formulae expressing the semantic constraints depend only on the underlying language not on the schema itself, the inference rules C1 to C5 make use of the underlying schema. So when we show the correctness of the inference rules, we take the underlying schema into account.

Theorem 4.9 (Soundness of the Inference Rules C1 to C5)
*The inference rules C1 to C5 are sound, i. e., for a set $\Upsilon \cup \{v\}$ of class inclusion constraints and onto constraints over a schema $D$: if $v \in \Upsilon^{+_D}$, then every instance of the schema $D$ satisfying $\Upsilon$ satisfies $v$,* $\mathrm{sat}(D \cup \Upsilon) \subset \mathrm{sat}(D \cup \{v\})$.

Proof. We show this by using the proof theory provided by F-logic. Admittedly, the use of the proof theory seems exaggerated, because the correctness of the inference rules seems to manifest itself semantically, but the proof theory helped the author in his efforts to develop the inference rules.

The proof goes now as follows. We show for all inference rules that the antecedents logically entail the conclusions. But the antecedents for the rules C2 (subclass inclusion), C3 (signature inclusion) and C5 (domain relaxation) comprehend parts of the class hierarchy giving and signature declaring components of a schema. So we show that if the constraint $v$ is derivable from the set $\Upsilon$ of constraints, then

$$D \cup \Upsilon \models v \tag{4.1}$$

holds. But now every completion of an instance under the schema $D \cup \Upsilon$ is an H-model of the set $D \cup \Upsilon$, and hence because of (4.1) an H-model of $v$. Therefore every instance of the schema $D \cup \Upsilon$ is also an instance of the schema $D \cup \{v\}$.

We need for this proof only a subset of the F-logic inference rules, namely the rules *resolution* and *subclass inclusion*. For this reason we do not introduce all F-logic inference rules only the two mentioned above. We merely sketch how the two F-logic inference rules work.

The symbols $L$ and $L'$ are used to denote positive literals, $C$ and $C'$ denote clauses, and $P$, $Q$, $Q'$, $R'$ denote id-terms. Resolution looks very much like resolution in classical logic.

Resolution: Let $\neg L \vee C$ and $L' \vee C'$ be a pair of clauses standardised apart (without common variables). Then the resolution rule is

$$\frac{\neg L \vee C, L' \vee C', \Theta = \mathrm{mgu}_{\sqsubseteq}(L, L')}{\Theta(C \vee C')} \quad .$$

In this rule $\Theta$ is the *most general unifier* of the literal $L$ into the literal $L'$.

Subclass inclusion: Let $(P{:}Q) \vee C$ and $(Q'{::}R') \vee C'$ be a pair of clauses standardised apart. Then the subclass inclusion rule says:

$$\frac{(P{:}Q) \vee C, (Q'{::}R') \vee C', \Theta = \mathrm{mgu}_{\sqsubseteq}(Q, Q')}{\Theta((P{:}R') \vee C \vee C')}$$

With the F-logic inference rules at hand, we can proceed nearly as in classical logic. Therefore we have to transform the formulae into clausal normal form. To obtain the clausal normal form of the onto constraint formula

$$\forall V \exists O (O : c[m \rightarrow V] \wedge c[m \Rightarrow ()] \longleftarrow V : c_m) \ ,$$

we skolemise it. So we get

$$f(V) : c[m \rightarrow V] \land c[m \Rightarrow ()] \longleftarrow V : c_m \ .$$

Then we bring the skolemised formula into conjunctive normal form, by first replacing the implication with its definition as disjunction,

$$\big(f(V) : c[m \rightarrow V] \land c[m \Rightarrow ()]\big) \lor \neg V : c_m \ ,$$

and then applying the distributive law and simultaneously breaking the first object molecule into its constituent atoms,

$$\big(f(V) : c \lor \neg V : c_m\big) \land \big(f(V)[m \rightarrow V] \lor \neg V : c_m\big) \land \big(c[m \Rightarrow ()] \lor \neg V : c_m\big) \ .$$

The conjunctive normal form of class inclusion constraint formulae is similarly obtained. It is

$$O : c' \lor \neg O : c$$

for a class inclusion constraint $c \subset c'$. Inference rule "signature inclusion" contains a signature atom in its antecedent. The way to capture this signature atom is to make use of the well-typedness axioms. In this case it is the axiom

$$\forall O \, \forall M \, \forall V \, \forall C \, \forall W (V : W \ \longleftarrow \ C[M \Rightarrow W] \land O[M \rightarrow V] \land O : C) \ .$$

Its conjunctive normal form is

$$V : W \lor \neg C[M \Rightarrow W] \lor \neg O[M \rightarrow V] \lor \neg O : C \ .$$

Now we give the refutations for all inference rules. In some refutation steps we implicitly make the clauses standardised apart without mentioning this explicitly.

| | | | |
|---|---|---|---|
| C1: | i. | $O : c' \lor \neg O : c$ | from $c \subset c'$ |
| | ii. | $O : c'' \lor \neg O : c'$ | from $c' \subset c''$ |
| | iii. | $\neg a : c''$ | from $c \subset c''$ |
| | iv. | $a : c$ | ” |
| | v. | $O : c'' \lor \neg O : c$ | by resolving (i) and (ii) |
| | vi. | $\neg a : c$ | by resolving (iii) with (v); $\Theta = \{O \backslash a\}$ |
| | vii. | $\square$ | by resolving (iv) with (vi) |

| | | | |
|---|---|---|---|
| C2: | i. | $c :: c'$ | |
| | ii. | $\neg a : c'$ | from $c \subset c'$ |
| | iii. | $a : c$ | ” |
| | iv. | $a : c'$ | by the rule of subclass inclusion, using (i) and (iii) |
| | v. | $\square$ | by resolving (ii) and (iv) |

C3: We witness in this refutation for the first time the influence of axioms inherent in a schema. In this case it is a part of the well-typedness axioms.

| | | |
|---|---|---|
| i. | $f(V) : d \vee \neg V : c$ | from $d\{m\|c\}$ |
| ii. | $f(V)[m \rightarrow V] \vee \neg V : c$ | " |
| iii. | $d[m \Rightarrow ()] \vee \neg V : c$ | " |
| iv. | $d[m \Rightarrow c']$ | |
| v. | $V : W \vee \neg C[M \Rightarrow W] \vee \neg O[M \rightarrow V] \vee \neg O : C$ | the well-typedness axiom |
| vi. | $\neg a : c'$ | from $c \subset c'$ |
| vii. | $a : c$ | " |
| viii. | $V : c' \vee \neg O[m \rightarrow V] \vee \neg O : d$ | by resolving (iv) and (v); $\Theta = \{C\backslash d, M\backslash m, W\backslash c'\}$ |
| ix. | $V : c' \vee \neg f(V) : d \vee \neg V : c$ | by resolving (ii) and (viii); $\Theta = \{O\backslash f(V)\}$ |
| x. | $V : c' \vee \neg V : c$ | by resolving (i) and (ix) |
| xi. | $\neg a : c$ | by resolving (vi) and (x); $\Theta = \{V\backslash a\}$ |
| xii. | $\Box$ | by resolving (vii) and (xi) |

C4:

| | | |
|---|---|---|
| i. | $O : c' \vee \neg O : c$ | from $c \subset c'$ |
| ii. | $f(V) : d \vee \neg V : c'$ | from $d\{m\|c'\}$ |
| iii. | $f(V)[m \rightarrow V] \vee \neg V : c'$ | " |
| iv. | $d[m \Rightarrow ()] \vee \neg V : c'$ | " |
| v. | $\neg O : d \vee \neg O[m \rightarrow a] \vee \neg d[m \Rightarrow ()]$ | from $d\{m\|c\}$ |
| vi. | $a : c$ | " |
| vii. | $a : c'$ | by resolving (i) with (vi); $\Theta = \{O\backslash a\}$ |
| viii. | $f(a) : d$ | by resolving (ii) and (vii); $\Theta = \{V\backslash a\}$ |
| ix. | $f(a)[m \rightarrow a]$ | by resolving (iii) and (vii); $\Theta = \{V\backslash a\}$ |
| x. | $d[m \Rightarrow ()]$ | by resolving (iv) and (vii); $\Theta = \{V\backslash a\}$ |
| xi. | $\neg f(a)[m \rightarrow a] \vee \neg d[m \Rightarrow ()]$ | by resolving (v) with (viii); $\Theta = \{O\backslash f(a)\}$ |
| xii. | $\neg d[m \Rightarrow ()]$ | by resolving (ix) and (xi) |
| xiii. | $\Box$ | by resolving (x) and (xii) |

C5:

| | | |
|---|---|---|
| i. | $O : c' \vee \neg O : c$ | from $c \subset c'$ |
| ii. | $f(V) : c \vee \neg V : d$ | from $c\{m\|d\}$ |
| iii. | $f(V)[m \rightarrow V] \vee \neg V : d$ | " |

| | | |
|---|---|---|
| iv. | $c[m \Rightarrow ()] \vee \neg V : d$ | ” |
| v. | $c'[m \Rightarrow ()]$ | from $m \in \mathrm{Attr}_D(c')$ |
| vi. | $\neg O : c' \vee \neg O[m \rightarrow a] \vee$ $\neg c'[m \Rightarrow ()]$ | from $c'\{m\|d\}$ |
| vii. | $a : d$ | ” |
| viii. | $f(a) : c$ | by resolving (ii) with (vii); $\Theta = \{V\backslash a\}$ |
| ix. | $f(a) : c'$ | by resolving (i) with (viii); $\Theta = \{O\backslash f(a)\}$ |
| x. | $f(a)[m \rightarrow a]$ | by resolving (iii) with (vii); $\Theta = \{V\backslash a\}$ |
| xi. | $\neg f(a)[m \rightarrow a] \vee \neg c'[m \Rightarrow ()]$ | by resolving (vi) and (ix); $\Theta = \{O\backslash f(a)\}$ |
| xii. | $\neg c'[m \Rightarrow ()]$ | by resolving (x) and (xi) |
| xiii. | $\square$ | by resolving (v) and (xii) |

$\square$

We demonstrate how the inference rules work by means of an example, which serves as a running example throughout the remainder of this section, Sections 4.2 and 4.3. The purpose of this example is to convey an intuition for the technicalities involved in the proof of the completeness of the inference rules for class inclusion constraints and onto constraints, therefore we do not come up with an example that offers an interpretation in the real world rather it uses class names and attribute names like $e$, $e'$, $j$ and alike.

Example 4.10

*Before we demonstrate how the inference rules do their job, we present a schema and then show how we can derive from a given set of semantic constraints, which are in this case only onto constraints, new semantic constraints, which an instance of the given schema has to satisfy as well due to the correctness of the inference rules (Theor. 4.9).*

*The schema $T$ we use has the components*

$$\begin{aligned}
\mathrm{CLASS}_T &= \{e, e', f, f', f'', g, h\} \ , \\
\mathrm{METH}_T &= \{j, k, n, p\} \ , \\
\mathrm{HIER}_T &= \{e{::}e', f''{::}f'\} \ , \\
\mathrm{SIG}_T &= \{e[j \Rightarrow f], e'[k \Rightarrow h], \\
&\qquad f''[p \Rightarrow e], \\
&\qquad h[n \Rightarrow g], \\
&\qquad g[k \Rightarrow h]\} \ , \ \textit{and} \\
\mathrm{SC}_T &= \mathrm{AX} \ .
\end{aligned}$$

*This schema is represented in Fig. 4.1.*

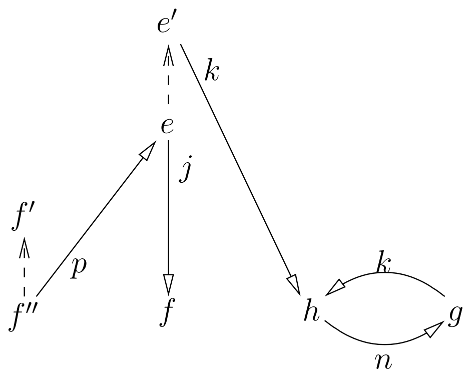
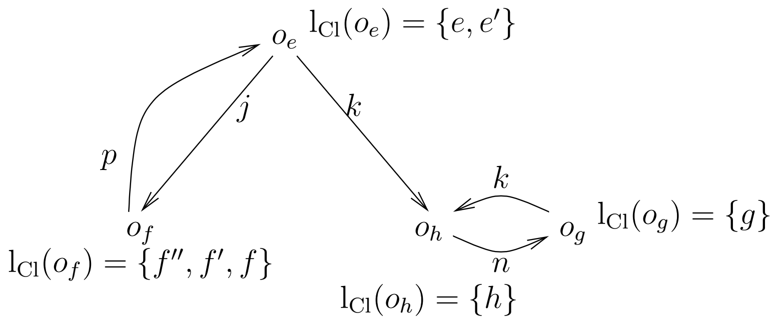


Figure 4.1: The schema $T$ in Exam. 4.10

*An instance of this schema $T$ is*

$$
\begin{aligned}
t = \langle\ &\{o_e{:}e, o_f{:}f'', o_f{:}f, o_h{:}h, o_g{:}g\} \mid \\
&\{o_e[j \rightarrow o_f; k \rightarrow o_h], \\
&\ \ o_f[p \rightarrow o_e], \\
&\ \ o_h[n \rightarrow o_g], \\
&\ \ o_g[k \rightarrow o_h]\}\ \rangle \quad .
\end{aligned}
$$

*This instance $t$ is depicted in Fig. 4.2.*



$o_e$
$\mathrm{l_{Cl}}(o_e) = \{e, e'\}$
j
k
p
k
$o_f$
$\mathrm{l_{Cl}}(o_f) = \{f'', f', f\}$
$o_h$
n
$o_g$
$\mathrm{l_{Cl}}(o_g) = \{g\}$
$\mathrm{l_{Cl}}(o_h) = \{h\}$

Figure 4.2: The instance $t$ of the schema $T$ in Exam. 4.10

*The set of semantic constraints used in this example to exhibit how the inference rules for class inclusion constraints and onto constraints operate is the set $\Omega$ of onto constraints over the schema $T$ with*

$$
\begin{aligned}
\Omega = \{&e\{j|f'\}, e\{k|h\}, \\
&f''\{p|e\}\} \quad .
\end{aligned}
$$

*For this set $\Omega$ of onto constraints we give an example for each of the inference rules C1* *to C5.*

$$\left.\begin{array}{r}\left.\begin{array}{r} e[j \Rightarrow f] \\ e\{j|f'\}\end{array}\right\}\overset{\text{C3}}{\vdash_T} f' \subset f \\ \left. f''{::}f' \right\}\overset{\text{C2}}{\vdash_T} f'' \subset f'\end{array}\right\}\overset{\text{C1}}{\vdash_T} f'' \subset f$$

$$\left.\begin{array}{r} e\{j|f'\} \\ f'' \subset f'\end{array}\right\}\overset{\text{C4}}{\vdash_T} e\{j|f''\}$$

$$\left.\begin{array}{r} e\{k|h\} \\ e \subset e' \\ e'[k \Rightarrow h]\end{array}\right\}\overset{\text{C5}}{\vdash_T} e'\{k|h\}$$

*To summarise, the set of all class inclusion constraints that can be derived from the set $\Omega$ of onto constraints under the schema $T$ is*

$$\begin{array}{l}\{e \subset e, e' \subset e', f'' \subset f'', f' \subset f', f \subset f, h \subset h, g \subset g, \\ \ e \subset e', f'' \subset f', f'' \subset f, f' \subset f\} \ \text{, and}\end{array}$$

*the set of all onto constraints that can be derived from the set $\Omega$ of onto constraints under the schema $T$ is*

$$\begin{array}{l}\{e\{j|f'\}, e\{j|f''\}, \\ \ e\{k|h\}, e'\{k|h\}, \\ \ f''\{p|e\}\} \ .\end{array}$$

*The instance $t$ of the schema $T$ is also an instance of the schema $T \cup \Omega$, and hence due to Theor. 4.9 an instance of the schema $T \cup \Omega^{+_T}$.*

*In the sequel we often revert to the schema $T' := T \cup \Omega$.*

An aspect where our data model differs from the relational data model is that we allow infinite instances. One of the reasons for the infinity is that the inference rules $C1$ to $C5$ are no longer complete for only finite instances ($\|\text{obj}(f)\| \in \mathbb{N}$). The other reason is discussed when we delineate the proof of the completeness.

EXAMPLE 4.11

*In this example we do not resort to the schema $T$ in Exam. 4.10 because if we had done so, we would have over-freighted that example. Instead we use a schema $D$, which includes the signature atom*

$$c[m \Rightarrow e] \tag{4.2}$$

*and use the onto constraints*

$$c\{m|d\} \tag{4.3}$$

*and*

$$d\{f|e\} \tag{4.4}$$

*over the schema $D$. From (4.2) and (4.3), we derive the class inclusion constraint $d \subset e$ by the inference rule C3 (signature inclusion). From (4.4), we conclude that the cardinality of the set of objects of the class $d$ is greater than or equal to the cardinality of the set of objects of the class $e$ in every instance $f$ of the schema $D$. Consequently,* *in every finite instance of the schema $D$, every object of the class $d$ is also an object of the class $e$ and vice versa, and so the onto constraint*

$$c\{m|e\} \tag{4.5}$$

*holds for every finite instance of the schema $D$.*

*An infinite instance of the schema $D$ that satisfies the onto constraints (4.3) and (4.4) but not the onto constraint (4.5) is represented in Tab. 4.6. The table merely introduces a creation scheme and uses "syntactic sugar". We read a "molecule" like $y'{:}c[m \to y{:}\{d,e\}]$ as follows. The is-a assertion $y'{:}c$ retains its known meaning. The expression $y{:}\{d,e\}$ stands for the two is-a assertions $y{:}d$ and $y{:}e$.*

| | |
|---|---|
| $y'{:}c[m \to y{:}\{d,e\}]$ | $y{:}d[f \to \mathbf{z}{:}e]$ |
| $x'{:}c[m \to x{:}\{d,e\}]$ | $x{:}d[f \to y{:}e]$ |
| $w'{:}c[m \to w{:}\{d,e\}]$ | $w{:}d[f \to x{:}e]$ |
| $\vdots$ | $\vdots$ |

Table 4.6: The creation scheme for an infinite instance in Exam. 4.11

*The first column in the creation scheme defines objects ($\{y', x', w', \ldots\}$) of the class $c$ along with their values for the attribute $m$ ($\{y, x, w, \ldots\}$), which are elements of the class $d$ as well as of the class $e$, thereby satisfying the well-typedness axioms activated by the signature atom* (4.2) *and satisfying the onto constraint* (4.3).

*The second column defines objects ($\{y, x, w, \ldots\}$) of the class $d$, which are already known to be elements of the class $d$, and their values for the attribute $f$ ($\{\mathbf{z}, y, x, \ldots\}$), which are elements of the class $e$, thereby satisfying the onto constraint* (4.4). *We detect in the first row and in the second column the object $\mathbf{z}$, which is an element of the class $e$, but which is never referenced by an object of the class $c$ via the attribute $m$. So the object $\mathbf{z}$ is a witness that the onto constraint* (4.5) *does not hold in this infinite instance.*

A result with respect to the inference rules C1 to C5 is that the inference rules are complete if we regard only class inclusion constraints alone. In this case we only use the inference rules C1 and C2 since it is not possible to derive onto constraints from a given set of class inclusion constraints, and so the rules C3 to C5 do not apply.

Lemma 4.12 (Completeness of C1 to C2 for CICs)
*Let $\Lambda \cup \{c \subset c'\}$ be a set of class inclusion constraints over a schema $D$ with only the set* AX *of axioms as semantic constraints,* $\mathrm{SC}_D = \mathrm{AX}$. *Then if the inclusion* $\mathrm{sat}(D \cup \Lambda) \subset \mathrm{sat}(D \cup \{c \subset c'\})$ *holds, we can derive the class inclusion constraint $c \subset c'$ from the set $\Lambda$ under the schema $D$, $\Lambda \vdash_D c \subset c'$.*

Proof. We conduct this proof by contraposition, i. e., we assume conversely that $\Lambda \not\vdash_D c \subset c'$ holds and show then that there exists an extension $f$ with $f \in \mathrm{sat}(D \cup \Lambda)$ and $f \notin \mathrm{sat}(D \cup \{c \subset c'\})$.

We construct the extension of the arbitrary schema $D$ satisfying the axioms AX and the set $\Lambda$ of class inclusion constraints but not the class inclusion constraint $c \subset c'$. This extension $f$ consists of two objects $o$ and $o'$ where the object $o$ is element of all classes $d$, for which we can derive $\Lambda \vdash_D c \subset d$, and the object $o'$ is element of all classes. The object $o'$ is the value of object $o$ for all attributes declared on all classes $d$, for which we can derive $\Lambda \vdash_D c \subset d$, and the value of the object $o'$ for all attributes for all classes, because the object $o'$ is element of all classes.

$$\begin{aligned} f := \langle\ & \{o{:}d \mid \Lambda \vdash_D c \subset d\} \cup \{o'{:}d \mid d \in \mathrm{CLASS}_D\} | \\ & \{o[m \rightarrow o'] \mid m \in \textstyle\bigcup_{d \in \{d' \mid \Lambda \vdash_D c \subset d'\}} \mathrm{Attr}_D(d)\} \cup \\ & \{o'[m \rightarrow o'] \mid m \in \mathrm{Attr}_D\}\ \rangle \end{aligned}$$

A sketch of the extension $f$ is outlined in Fig. 4.3.

$$\begin{tikzcd} o \arrow[r] & o' \arrow[loop right] & \mathrm{l}_{\mathrm{Cl}}(o') = \mathrm{CLASS}_D \\ \mathrm{l}_{\mathrm{Cl}}(o) = \{c, \ldots\} & & \end{tikzcd}$$

Figure 4.3: Sketch for the extension in the proof of Lem. 4.12

The extension $f$ satisfies the unique name axioms, well-typedness axioms and definedness axioms. So it is an instance of the schema $D$.

The objects $o$ and $o'$ satisfy the set of class inclusion constraints $\Lambda$ by definition. Therefore the extension $f$ is also an instance of the schema $D \cup \Lambda$.

Since the hypothesis $\Lambda \not\vdash_D c \subset c'$ holds, we know due to the inference rule C2 that $\mathrm{HIER}_D \not\models c{::}c'$ holds, which means $\mathrm{compl}_D(f) \not\models c{::}c'$ holds by the proof theory of F-logic. But then the object $o$ is not an element of the class $c'$. Therefore the extension $f$ is not an instance of the schema $D \cup \{c \subset c'\}$.

□

Before we conclude this section with an outline of how we prove the completeness of the inference rules C1 to C5 when both kinds of semantic constraints, class inclusion constraints and onto constraints, are present, we show the problem $\Upsilon \vdash_D \upsilon$ is decidable for a finite set $\Upsilon \cup \{\upsilon\}$ of class inclusion constraints and onto constraints.

THEOREM 4.13 (DECIDABILITY OF THE DERIVABILITY PROBLEM FOR C1 TO C5) *Let $\Upsilon \cup \{\upsilon\}$ be a finite set of class inclusion constraints and onto constraints over a schema $D$. The problem $\Upsilon \vdash_D \upsilon$ is decidable.*

PROOF. Because of the finiteness of most components in a schema our syntactic material is restricted. Due to Theor. 3.4 the set of all non-trivially implied class is-a assertions that can be deduced from the set $\mathrm{HIER}_D$, which is finite, can be effectively computed as well as the set of all implied signature atoms that can be deduced from the set $\mathrm{HIER}_D \cup \mathrm{SIG}_D$, which is finite again. Both deduced sets are finite, and so the set of all constraints derivable from $\Upsilon$ by the inference rules C1 to C5 is finite with an upper bound, which is set by the number of all class inclusion constraints and onto constraints over the schema $D$, and can be effectively computed. $\square$

The proof of the completeness of the inference rules for class inclusion constraints and onto constraints when both kinds of semantic constraints are present is harder to show. Therefore we give first an outline of how we conduct this proof. We show for a set $\Upsilon \cup \{\upsilon\}$ of class inclusion constraints and onto constraints and for a schema $D$ with only the set AX of axioms as semantic constraints, $\mathrm{SC}_D = \mathrm{AX}$, that if every instance of the schema $D \cup \Upsilon$ is an instance of the schema $D \cup \{\upsilon\}$, then $\Upsilon \vdash_D \upsilon$ can be derived. But instead of showing this assertion directly, we conduct the proof by contraposition. So we assume conversely that we cannot derive the constraint $\upsilon$ from the set $\Upsilon$, $\Upsilon \not\vdash_D \upsilon$. Then we construct an instance $f$ of the schema $D$ such that $f$ is

- an instance of the schema $D \cup \Upsilon$,
- but not an instance of the schema $D \cup \{\upsilon\}$.

Such an instance has to be constructed with great diligence in order to meet the demands above. The idea is to start first with an extension that satisfies at least the set AX of axioms and later on to amend this extension to satisfy the set $\Upsilon$ of constraints by upholding the invariant that the extension does not satisfy the constraint $\upsilon$. We call such a prototype instance an *S-tree* (Def. 4.30).

The way to construct such an S-tree for a schema $D$ and a set $\Upsilon \cup \{\upsilon\}$ of constraints where $\upsilon$ is of the form $d\{m|c\}$ or $c \subset d$ is to take an object $r$ of the class $c$ in both cases as starting point. As mentioned before we want the constructed extension to satisfy first the set AX of axioms. Applying this insight to our extension, consisting so far only of the object $r$, means that the definedness axioms DEF have to be satisfied. Thus we have to introduce appropriate values for the attributes of the object $r$. According to the well-typedness axioms, these attribute values are objects of some classes, which in turn have signatures declared on them. So again as for the object $r$, we have to introduce attribute values to satisfy the definedness axioms. This process where we always introduce new objects to avoid conflicts with the unique name axioms has to be iterated until all axioms are satisfied (Lem. 4.32). The outcome of this iteration has the form of a tree with the object $r$ as root, hence the name *S-Tree*. If the declaration of signatures in the schema $D$ contains a cycle, e. g. declarations like $a[m \Rightarrow b]$ and $b[n \Rightarrow a]$, the tree becomes infinite. Therefore we allow infinite extensions and instances.

A node of this tree is an object of the extension, and an edge $o \xrightarrow{m} o'$ with the label $m$ represents that the object $o'$ is the value for the attribute $m$ of the object $o$. This edge has been introduced, because the object $o$ is an element of some class $d$, for which a signature $d[m \Rightarrow d']$ is declared. Looking from a different perspective, we discover that knowing the schema is sufficient to describe an *attribute value path* in an S-tree. For example if a schema contains the signature atoms $d[m \Rightarrow d']$ and $d'[n \Rightarrow g]$, they might describe the attribute value path $o \xrightarrow{m} p \xrightarrow{n} q$ in the S-tree. We use “might” in the preceding sentence, because the signature atoms only possess the potential to describe an attribute value path. If and only if the object $o$ is element of the class $d$, the attribute value path $o \xrightarrow{m} p \xrightarrow{n} q$ must exist. Instead of using signature atoms, we condense the imparted information of these signature atoms into the notion of a path description, $.m.n$ in this example. By means of this path description, we can describe the attribute value path $o \xrightarrow{m} p \xrightarrow{n} q$.

Because of the definedness axioms and the scalarity of scalar methods, hence attributes, a path description appropriately applied to an object describes exactly one attribute value path, and we therefore denote path descriptions as *path functions* (Def. 4.23).

Coming back to the extension, known to be an S-tree and satisfying the set AX of axioms, we recall our immediate goal: we want to amend the S-tree in such a way that the set $\Upsilon$ of constraints is satisfied. When an onto constraint $d\{m|c\}$ is violated in the extension there exists an object $o$, element of the class $c$, that is not referenced by an object of the class $d$ via the attribute $m$. To remedy the violation, we insert a new object $o'$ into the extension, make the object $o'$ an element of the class $d$, and stipulate the object $o$ as value for the attribute $m$ of the object $o'$.

We soon discover we have to bestow some care making the insertion—we have to ensure that the set AX of axioms is satisfied again. So instead of inserting only one single object $o'$, we insert an S-tree with root $o'$, which we prune by cutting off the branch starting with the attribute in the violated constraint, which is the attribute $m$ in our example. We call such a pruned S-tree *pruned-S-tree* (Def. 4.33).

We simultaneously insert pruned-S-trees for all objects violating onto constraints and call this operation extrev (Def. 4.37), which upholds the invariant that if an extension satisfies the set AX of axioms, then extrev$(f)$ satisfies the set AX of axioms (Lem. 4.39).

Applying the operation extrev on an S-tree does not yield an extension that looks like a tree any longer. For example we assume that the root $r$ of an S-tree does not satisfy the onto constraint $d\{m|c\}$. To remedy this violation, we insert a new object $o$, make the object $o$ an element of the class $d$ and insert the edge $o \xrightarrow{m} r$ into the S-tree. But then we can no longer describe this edge by a path function starting at the root $r$. The edge $o \xrightarrow{m} r$ runs in the reverse direction when we look from the perspective of the root $r$. A fact that is covered by the concept of a *way description* (Def. 4.23), which is an extension of the concept of a path function to capture this reverse traversal of attribute value paths.

Iterating this process of introducing new objects produces then an extension that satisfies the onto constraints derivable from the set $\Upsilon$ of class inclusion constraints and onto constraints (Lem. 4.40).

To satisfy the class inclusion constraints derivable from this set but not the class inclusion constraint $v$, we follow a different strategy, which ensures that the class inclusion constraints are satisfied from the outset but not the class inclusion constraint $v$. So when we begin with the root of the initial S-tree, we ensure that the root is element of all classes required by the set $\Upsilon$ of constraints but of no other classes. As we do not know at this stage whether the inference rules are complete, we obtain the knowledge about the required class memberships by derivation using the inference rules, which are at least correct (Theor. 4.9). The iteration process does not alter the class membership of any object, so all objects satisfy all class inclusion constraints derivable from the set $\Upsilon$ of constraints.

Finally, we know that the iteration process generates an output that satisfies both the set AX of axioms and the set $\Upsilon$ of constraints but not the constraint $\upsilon$, and thus we prove the completeness of the inference rules C1 to C5 (Theor. 4.41).

## 4.2 Schema and Instance Navigation

As mentioned before in the proof outline, we construct an extension that satisfies all derivable class inclusion constraints, which influence the class membership of the objects in this extension. Our starting point in the construction is a class $c$ occurring in either a class inclusion constraint $c \subset d$ or an onto constraint $d\{m|c\}$. Our first object in the construction, the root of the S-tree, becomes a member of this class $c$ and all other classes $c'$ with $c \subset c'$ according to what we can derive from the set $\Upsilon$ of constraints. We denote this set of classes the *folder* of the class $c$. A folder of a class $c$ specifies all classes an object of the class $c$ must be a member of.

Definition 4.14 (Folder)
*Let $D$ be a schema. We define the* folder *of a class $c \in \mathrm{CLASS}_D$ as*

$$\mathrm{Folder}_D(c) := \{d \mid \mathrm{SC}_D \backslash \mathrm{AX} \vdash_D c \subset d\} \ .$$

Note that the set $\mathrm{SC}_D$ of semantic constraints consists in this section only of axioms, class inclusion constraints and onto constraints; but we do not make this fact explicit in the definition because we want to reuse the definition in other sections where the semantic constraints are not exclusively restricted to axioms, class inclusion constraints and onto constraints.

Example 4.15
*In Exam. 4.10 the folder of the class $f''$ is* $\mathrm{Folder}_{T'}(f'') = \{f'', f', f\}$.

Having determined which classes the initial object has to be a member of, we have to find out about the class memberships of other objects included in the initial extension due to the definedness axioms and well-typedness axioms, and later on in the iterated extensions even due to unsatisfied onto constraints. For this task we employ *way descriptions*, which describe the attribute value ways in the extension. Way descriptions are an extension of path functions [Wed89, Wed92], which capture that we follow attribute value edges like $o \xrightarrow{m} o'$ not only in the forward direction from $o$ to $o'$ but also in the reverse direction from $o'$ to $o$. Syntactically, way descriptions are strings of attribute names separated by dots or minuses, where dots signify the forward traversal while minuses signify the backward traversal. In our first definition of *(arbitrary) way descriptions*, we arbitrarily string attribute names together.

Definition 4.16 ((Arbitrary) Way Description)
*The set of* (arbitrary) way descriptions $\mathrm{W}_{\mathrm{arb}}(D)$ *over a schema $D$ consists of all finite sequences of 'dot or minus separated' attribute names together with the* identity way

description *denoted as* .Id *where* Id $\notin \mathcal{V} \cup \mathcal{S}$.

$$\mathrm{W_{arb}}(D) := \{\pi_1 m_1 \cdots \pi_n m_n \mid n > 0 \ \text{ and } \ m_i \in \mathrm{Attr}_D, \pi_i \in \{., -\}, i \in \{1, \dots, n\}\} \\ \cup \{.\mathrm{Id}\}$$

*The length of the way description* .Id *is* 0, len(.Id) := 0. *The length of a way description* $w \equiv \pi_1 m_1 \cdots \pi_n m_n$ *is* $n$, $\mathrm{len}(w) := n$.

Example 4.17
*For the schema* $T$ *in Exam. 4.10 the following are arbitrary way descriptions over the schema* $T$*:*

$$\begin{array}{l} -j-k.p-n.k-k, \\ .\mathrm{Id}, \\ .j.k.p.n.k, \\ .k.n, \\ -n-k, \\ -k, \\ -k-p \ . \end{array}$$

A *path function* is a special way description with only dots as separating symbols, because a path function is intended to describe attribute value paths.

Definition 4.18 ((Arbitrary) Path Function)
*The set of* (arbitrary) path functions $\mathrm{P_{arb}}(D) \subset \mathrm{W_{arb}}(D)$ *over a schema* $D$ *consists of all arbitrary way descriptions with only 'dots' occurring in them.*

Example 4.19
*The arbitrary way descriptions*

$$\begin{array}{l} .\mathrm{Id}, \\ .j.k.p.n.k, \\ .k.n \end{array}$$

*are arbitrary path functions over the schema* $T$ *in Exam. 4.10.*

The set of arbitrary way descriptions is too large, because way descriptions describe attribute value ways that do not appear in any instance of the underlying schema. In addition, there are arbitrary way descriptions we are not interested in. These way descriptions take on the form $\cdots .m-m \cdots$ or $\cdots -m.m \cdots$, which we exclude by means of the concatenation of arbitrary way descriptions.

Definition 4.20 (Concatenation of Arbitrary Way Descriptions)
*Let* $w_1, w_2 \in \mathrm{W_{arb}}(D)$ *be two way descriptions over a schema* $D$. *Then the concatenation, written* $w_1 \circ w_2$, *is defined as*

$$w_1 \circ w_2 := \begin{cases} .\mathrm{Id} & \text{if } w_1 = w_2 = .\mathrm{Id}, \text{ or } w_1 = \pi m,\ w_2 = \bar{\pi} m \\ w_1 & \text{if } w_1 \neq .\mathrm{Id},\ w_2 = .\mathrm{Id} \\ w_2 & \text{if } w_1 = .\mathrm{Id},\ w_2 \neq .\mathrm{Id} \\ w_1' & \text{if } w_1 = w_1' \pi m,\ w_2 = \bar{\pi} m \\ w_2' & \text{if } w_1 = \pi m,\ w_2 = \bar{\pi} m w_2' \\ w_1' \circ w_2' & \text{if } w_1 = w_1' \pi m,\ w_2 = \bar{\pi} m w_2' \\ w_1 w_2 & \text{otherwise} \end{cases} ,$$

*where, if $\pi = .$ or $\pi = -$, then $\bar{\pi} = -$ or $\bar{\pi} = .$, respectively.*

Example 4.21
*We give examples for the concatenation of arbitrary way descriptions for the arbitrary way descriptions in Exam. 4.17.*

$$\begin{aligned}
&.\mathrm{Id} \circ .\mathrm{Id} = .\mathrm{Id}\\
&-j-k.p-n.k-k \circ .\mathrm{Id} = -j-k.p-n.k-k\\
&.\mathrm{Id} \circ -j-k.p-n.k-k = -j-k.p-n.k-k\\
&.k.n \circ -n-k = .\mathrm{Id}\\
&-n-k \circ .k.n = .\mathrm{Id}\\
&.k.n \circ .k.n = .k.n.k.n
\end{aligned}$$

The concatenation of arbitrary way descriptions is associative as the next lemma indicates, for instance

$$(.k.n \circ -n-k) \circ .k.n = .\mathrm{Id} \circ .k.n = .k.n = .k.n \circ .\mathrm{Id} = .k.n \circ (-n-k \circ .k.n) \ .$$

Lemma 4.22 (Associativity of Arbitrary Way Descriptions)
*Let $w_1, w_2, w_3 \in \mathrm{W}_{\mathrm{arb}}(D)$ be three way descriptions over a schema $D$. The concatenation of way descriptions is associative,*

$$(w_1 \circ w_2) \circ w_3 = w_1 \circ (w_2 \circ w_3) \ .$$

Proof. The proof of this lemma includes some technical subtleties but is obvious.

□

Our objective is to use only those way descriptions, *well-formed way descriptions*, that always describe an attribute value way starting at an object in certain instances. These instances are minimal in the sense that apart from a given set of objects they only contain objects whose existence is demanded by the semantic constraints and they form a directed acyclic graph. Then we intend to determine the compulsory class memberships of the object reached by the attribute value way. In fact there is a close correspondence between the compulsory class memberships of such an object and the definition of way descriptions.

The definition of a well-formed way description starts with the shortest possible well-formed way description, .Id, and then considers longer descriptions by distinguishing the prolongation of an already existing well-formed way description by an attribute either in the forward or the reverse direction.

We simultaneously define the *domain* and the *range* of a well-formed way description. The former is the set of classes from which a way description may start, i. e. a well-formed way description always describes an attribute value way if the starting point is an object that is element of a class in the domain of the way description. The latter is a set of classes an end node of an attribute value way must be a member of provided the start node is element of a class in the domain of the way description and the extension under consideration obeys certain restrictions.

Definition 4.23 (Well-Formed Way Description)

1. *The set of* well-formed way descriptions $\mathrm{W_{wf}}(D) \subset \mathrm{W_{arb}}(D)$ *over a schema $D$ is the smallest set such that*

   - $.\mathrm{Id} \in \mathrm{W_{wf}}(D)$, *where*
     - $\mathrm{Dom}_D(.\mathrm{Id}) := \mathrm{CLASS}_D$, *and*
     - $\mathrm{Ran}_D(c, .\mathrm{Id}) := \mathrm{Folder}_D(c)$ *for all* $c \in \mathrm{CLASS}_D$,
   - *if* $w \in \mathrm{W_{wf}}(D)$ *is a well-formed way description with a domain class* $c \in \mathrm{Dom}_D(w)$ *such that* $m \in \bigcup_{d \in \mathrm{Ran}_D(c,w)} \mathrm{Attr}_D(d)$ *for some attribute $m$ and* $\mathrm{len}(w) < \mathrm{len}(w \circ .m)$, *then* $w \circ .m \in \mathrm{W_{wf}}(D)$, *where*

$$\mathrm{Dom}_D(w \circ .m) := \{e \in \mathrm{Dom}_D(w) \mid m \in \bigcup_{f \in \mathrm{Ran}_D(e,w)} \mathrm{Attr}_D(f)\} \ , \textit{and}$$

$$\begin{aligned}\mathrm{Ran}_D(e, w \circ .m) := \{h \in \mathrm{Folder}_D(g) \mid \textit{ex. } f\colon\ & f \in \mathrm{Ran}_D(e,w) \textit{ and} \\ & \mathrm{HIER}_D \cup \mathrm{SIG}_D \models f[m \Rightarrow g]\}\end{aligned}$$

$$\textit{for all } \ e \in \mathrm{Dom}_D(w \circ .m) \ ,$$

   - *if* $w \in \mathrm{W_{wf}}(D)$ *is a well-formed way description with a domain class* $c \in \mathrm{Dom}_D(w)$ *such that* $d\{m|f\} \in (\mathrm{SC}_D \backslash \mathrm{AX})^{+_D}$ *for some class* $f \in \mathrm{Ran}_D(c,w)$, *some attribute $m$, and some class $d$, and* $\mathrm{len}(w) < \mathrm{len}(w \circ -m)$, *then* $w \circ -m \in \mathrm{W_{wf}}(D)$, *where*

$$\begin{aligned}\mathrm{Dom}_D(w \circ -m) := \{e \in \mathrm{Dom}_D(w) \mid \textit{ex. } f, g\colon\ & f \in \mathrm{Ran}_D(e,w) \textit{ and} \\ & g\{m|f\} \in (\mathrm{SC}_D \backslash \mathrm{AX})^{+_D}\} \ , \textit{and} \\ \mathrm{Ran}_D(e, w \circ -m) := \{h \in \mathrm{Folder}_D(g) \mid \textit{ex. } f\colon\ & f \in \mathrm{Ran}_D(e,w) \textit{ and} \\ & g\{m|f\} \in (\mathrm{SC}_D \backslash \mathrm{AX})^{+_D}\}\end{aligned}$$

$$\textit{for all } e \in \mathrm{Dom}_D(w \circ -m) \ .$$

2. *For each class* $c \in \mathrm{CLASS}_D$ *we write* $\mathrm{WayDes}_D(c)$ *to denote all way descriptions* $w \in \mathrm{W_{wf}}(D)$ *where* $c \in \mathrm{Dom}_D(w)$, *i. e. all way descriptions starting at the class $c$.*

3. *The set* $\mathrm{PathFuncs}_D(c) \subset \mathrm{WayDes}_D(c)$ *denotes the set of all well-formed path functions starting at the class $c$.*

When we speak in the sequel of way descriptions (path functions), we always mean well-formed way descriptions (path functions) unless otherwise mentioned.

Example 4.24

*Some of the arbitrary way descriptions in Exam. 4.17 are well-formed way descriptions. First of all the arbitrary way description* $.\mathrm{Id}$ *is a well-formed way description with*

$$\begin{aligned}&\mathrm{Dom}_{T'}(.\mathrm{Id}) = \mathrm{CLASS}_{T'} = \{e, e', f'', f', f, h, g\} \quad , \ \textit{and} \\ &\mathrm{Ran}_{T'}(f'', .\mathrm{Id}) = \mathrm{Folder}_{T'}(f'') = \{f'', f', f\} \ .\end{aligned}$$

*Based on this knowledge we calculate for the way descriptions $.k.n$ and $-k-p$ their domains and some of their ranges.*

*We begin with the way description .k.n. Therefore we show first that .k is a well-formed way description and then determine its domain and range.*

*We know that the way description* $.\mathrm{Id}$ *is well-formed. Now for the class* $g \in \mathrm{Dom}_{T'}(.\mathrm{Id})$ *the range* $\mathrm{Ran}_{T'}(g, .\mathrm{Id})$ *of the way description* $.\mathrm{Id}$ *is* $\mathrm{Folder}_{T'}(g) = \{g\}$. *The set* $\mathrm{Attr}_{T'}(g)$ *of attributes for the class g is* $\{k\}$, *hence* $k \in \mathrm{Attr}_{T'}(g)$. *Finally, the inequation* $\mathrm{len}(.\mathrm{Id}) = 0 < 1 = \mathrm{len}(.k) = \mathrm{len}(.\mathrm{Id} \circ .k)$ *holds, and therefore the way description .k is well-formed with*

$$\begin{aligned}\mathrm{Dom}_{T'}(.k) &= \mathrm{Dom}_{T'}(.\mathrm{Id} \circ .k)\\ &= \{a \in \mathrm{Dom}_{T'}(.\mathrm{Id}) \mid k \in \bigcup_{b \in \mathrm{Ran}_{T'}(a, .\mathrm{Id})} \mathrm{Attr}_{T'}(b)\}\\ &= \{e, e', g\}\end{aligned}$$

*and*

$$\begin{aligned}\mathrm{Ran}_{T'}(g, .k) &= \mathrm{Ran}_{T'}(g, .\mathrm{Id} \circ .k)\\ &= \{c \in \mathrm{Folder}_{T'}(b) \mid \textit{ex. } a\colon a \in \mathrm{Ran}_{T'}(g, .\mathrm{Id}) \textit{ and}\\ &\qquad\qquad \mathrm{HIER}_{T'} \cup \mathrm{SIG}_{T'} \models a[k \Rightarrow b]\}\\ &= \{h\} \ .\end{aligned}$$

*Now we know* $g \in \mathrm{Dom}_{T'}(.k)$, $h \in \mathrm{Ran}_{T'}(g, .k)$, $n \in \mathrm{Attr}_{T'}(h)$ *and* $\mathrm{len}(.k) = 1 < 2 = \mathrm{len}(.k.n) = \mathrm{len}(.k \circ .n)$, *and thus the way description .k.n is well-formed with*

$$\begin{aligned}\mathrm{Dom}_{T'}(.k.n) &= \{e, e', g\} \quad \textit{and}\\ \mathrm{Ran}_{T'}(e, .k.n) = \mathrm{Ran}_{T'}(e', .k.n) &= \mathrm{Ran}_{T'}(g, .k.n) = \{g\} \ .\end{aligned}$$

*We exhibit how the definition can be used with way descriptions like* $-k$. *The way description* $-k$ *is well-formed, because* $h \in \mathrm{Dom}_{T'}(.\mathrm{Id})$, $h \in \mathrm{Ran}_{T'}(h, .\mathrm{Id}), e'\{k|h\} \in \Omega^{+_{T'}}$ *and* $\mathrm{len}(.\mathrm{Id}) < \mathrm{len}(-k)$ *holds. So the way description* $-k$ *is well-formed with*

$$\begin{aligned}\mathrm{Dom}_{T'}(-k) &= \{h\} \quad \textit{and}\\ \mathrm{Ran}_{T'}(h, -k) &= \{e, e'\} \ .\end{aligned}$$

When we concat two way descriptions such that the second is applicable to a class in the range of the first, we obtain a way description again, independent of the form of both way descriptions.

Lemma 4.25 (Concatenation of Way Descriptions)
*Let* $w, w' \in \mathrm{W}_{\mathrm{wf}}(D)$ *be two way descriptions over a schema D such that there exist classes c, c′ with* $c \in \mathrm{Dom}_D(w)$ *and* $c' \in \mathrm{Ran}_D(c, w) \cap \mathrm{Dom}_D(w')$. *Then* $w \circ w'$ *is a way description over the schema D,* $w \circ w' \in \mathrm{W}_{\mathrm{wf}}(D)$, *with* $c \in \mathrm{Dom}_D(w \circ w')$ *and* $\mathrm{Ran}_D(c', w') \subset \mathrm{Ran}_D(c, w \circ w')$.

Proof. We conduct the proof by induction on the length of the way description $w'$.

$\mathrm{len}(w') = 0$. Then $c \in \mathrm{Dom}_D(w) = \mathrm{Dom}_D(w \circ w')$ and

$$\mathrm{Ran}_D(c', w') \overset{\text{Def. 4.23}}{=} \mathrm{Folder}_D(c') \subset \mathrm{Ran}_D(c, w) = \mathrm{Ran}_D(c, w \circ w') \ .$$

The equation $\mathrm{Folder}_D(c') \subset \mathrm{Ran}_D(c, w)$ holds because of the hypothesis $c' \in \mathrm{Ran}_D(c, w)$, Def. 4.14 and Def. 4.23.

$\mathrm{len}(w') = n + 1$, $n \geq 0$. Then the way description $w'$ is of the form $w'' \circ .m$ or $w'' \circ -m$, and $w'' \in \mathrm{W}_{\mathrm{wf}}(D)$ is a way description over the schema $D$ with $c \in \mathrm{Dom}_D(w \circ w'')$ and $\mathrm{Ran}_D(c', w'') \subset \mathrm{Ran}_D(c, w \circ w'')$ according to the inductive assumption.

$w' = w'' \circ .m$. Then we distinguish two cases according to the form of the way description $w \circ w''$.

$w \circ w'' = x\pi y$ **with** $\pi \in \{., -\}$, $y \in \mathrm{METH}_D$, $\pi y \neq -m$. Because of $c' \in \mathrm{Dom}_D(w')$, $c' \in \mathrm{Dom}_D(w'')$ holds, and there exists a class $d \in \mathit{Ran}_D(c', w'')$ with $m \in \mathrm{Attr}_D(d)$. Consequently, $c \in \mathrm{Dom}_D(w \circ w'')$, $d \in \mathrm{Ran}_D(c, w \circ w'')$, $m \in \mathrm{Attr}_D(d)$ and $\mathrm{len}(w \circ w'') < \mathrm{len}((w \circ w'') \circ .m)$. Then by Def. 4.23, $c \in \mathrm{Dom}_D((w \circ w'') \circ .m)$ and $\mathrm{Ran}_D(c', w'' \circ .m) \subset \mathrm{Ran}_D(c, (w \circ w'') \circ .m)$.

$w \circ w'' = x{-}m$. Then $x = w \circ w'$ is a way description with $c \in \mathrm{Dom}_D(x)$, and $\mathrm{Ran}_D(c', w') \subset \mathrm{Ran}(c, x)$ by Def. 4.23 and by the fact that the way description $x$ is a prefix of the way description $w \circ w''$.

$w' = w'' \circ -m$. Then we distinguish two cases according to the form of the way description $w \circ w''$.

$w \circ w'' = x\pi y$ **with** $\pi \in \{., -\}$, $y \in \mathrm{METH}_D$, $\pi y \neq .m$. Because of $c' \in \mathrm{Dom}_D(w')$, $c' \in \mathrm{Dom}_D(w'')$ holds, and there exists a class $d \in \mathrm{Ran}_D(c', w'')$ with $e\{m|d\} \in (\mathrm{SC}_D \backslash \mathrm{AX})^{+_D}$. Consequently, $c \in \mathrm{Dom}_D(w \circ w'')$, $d \in \mathrm{Ran}_D(c, w \circ w'')$, $e\{m|d\} \in (\mathrm{SC}_D \backslash \mathrm{AX})^{+_D}$ and $\mathrm{len}(w \circ w'') < \mathrm{len}((w \circ w'') \circ -m)$. Then by Def. 4.23, $c \in \mathrm{Dom}_D((w \circ w'') \circ -m)$ and $\mathrm{Ran}_D(c', w'' \circ -m) \subset \mathrm{Ran}_D(c, (w \circ w'') \circ -m)$.

$w \circ w'' = x.m$. Then $x = w \circ w'$ is a way description with $c \in \mathrm{Dom}_D(x)$, and $\mathrm{Ran}_D(c', w') \subset \mathrm{Ran}(c, x)$ by Def. 4.23 and by the fact that the way description $x$ is a prefix of the way description $w \circ w''$.

$\square$

A well-formed way description might describe several attribute value ways when starting at the same object. Mostly we are interested in the end points of an attribute value way described by a well-formed way description.

Definition 4.26 (Application of Way Descriptions)
*Let $f$ be an instance of a schema $D$, $o \in \mathrm{obj}(f)$ be an object, $c \in \mathrm{l}_{\mathrm{Cl}}(o)$ be a class, $m \in \mathrm{Attr}_D$ be an attribute, and $w \in \mathrm{WayDes}_D(c)$ be a way description.*

- *If $w = .\mathrm{Id}$, then $o.w := \{o\}$.*

- *If $w = .m$, then $o.w := \{o'\}$ with* $\mathrm{ob}_f \models o[m \to o']$.
- *If $w = -m$, then $o.w := \{o' \mid \mathrm{ob}_f \models o'[m \to o]\}$.*
- *If $w = w'.m$, then $o.w := \{o'' \mid$ ex. $o'$: $o' \in o.w'$ and* $\mathrm{ob}_f \models o'[m \to o'']\}$.
- *If $w = w'-m$, then $o.w := \{o'' \mid$ ex. $o'$: $o' \in o.w'$ and* $\mathrm{ob}_f \models o''[m \to o']\}$.

Example 4.27
*The application of way descriptions $.k$, $.k.n$, $-k$, $-k-p$ and $-k-p-j$ to some objects in the extension $t$ in Exam. 4.10 yields the following results*

$$\begin{aligned} o_e..k &= \{o_h\} \ , \\ o_e..k.n. &= \{o_g\} \ , \\ o_h.-k &= \{o_e, o_g\} \ , \\ o_h.-k-p &= \{o_f\} \ \textit{and} \\ o_h.-k-p-j &= \{o_e\} \ . \end{aligned}$$

*The instance $t$ is definitely not an instance we are interested in, because the instance does not form a directed acyclic graph. So the application of the way description $-k$ on the object $o_h$ yields the set $\{o_e, o_g\}$, and the object $o_g$ is not a member of one of the classes in* $\mathrm{Ran}_{T'}(h, -k)$.

## 4.3 Completeness of OCs and CICs

As mentioned in the outline of the completeness proof, we use special extensions with certain qualities. These extensions satisfy the set AX of axioms and the set of class inclusion constraints derivable from the given set of class inclusion constraints and onto constraints.

There is a tight correspondence between the satisfaction of the axioms and the class inclusion constraints. When a class inclusion constraint demands that an object $o$ is element of a class $d$, this object $o$ delivers a value for an attribute declared on the class $d$ due to the definedness axioms. On the other hand if the object $o$ is the value for the attribute $k$ of an object $o'$, $o'[k \to o]$, the object $o'$ is element of the class $d'$ and the signature $d'[k \Rightarrow d]$ is declared, the object $o$ is a member of the class $d$ due the well-typedness axioms, and hence a member of all classes $e$ for which we derive $d \subset e$.

Our means to observe the accordance is to adorn each object in the special extensions with a label, a *way-label*. Each way-label is a way description and indicates that this object is reachable by an attribute value way described by the way-label. The attribute value way begins at a particular object in the extension called *root*, which is identical with the root of an S-tree mentioned in the proof outline. We denote the special extensions *extensions with labelling*, and a root[3] carries the way-label .Id. An extension with labelling is parameterised with an underlying schema with only the set AX of axioms as

[3]We allow several roots in an extension with labelling, a feature that we exploit in Sect. 4.5.

semantic constraints, a set of class inclusion constraints and onto constraints over the schema and a set of classes, which stipulate the set of classes a root is a member of.

Definition 4.28 (Instance (Extension) with Labelling)
*Let $f$ be an instance (extension) of a schema $D$ with only the set* AX *as semantic constraints,* $\mathrm{SC}_D = \mathrm{AX}$*,* $\Upsilon$ *be a set of class inclusion constraints and onto constraints over the schema $D$, and $C \subset \mathrm{CLASS}_D$ be a set of classes. The instance (extension) $f$ is an* instance (extension) with labelling based on the sets $C$ and $\Upsilon$*, if there is a* way-label $\mathrm{l_{Wf}}(o) \in \bigcup_{c\in C} \mathrm{WayDes}_{D\cup\Upsilon}(c)$ *for each object $o \in \mathrm{obj}(f)$ with the following properties*

1. $\mathrm{l_{Cl}}(o) = \bigcup_{c\in C\cap \mathrm{Dom}_{D\cup\Upsilon}(\mathrm{l_{Wf}}(o))} \mathrm{Ran}_{D\cup\Upsilon}(c, \mathrm{l_{Wf}}(o))$*, and*
2. *for all objects $o' \in \mathrm{obj}(f)$, if $\mathrm{ob}_f \models o[m \to o']$, then $\mathrm{l_{Wf}}(o) \circ .m = \mathrm{l_{Wf}}(o')$.*

*We call an object $o \in \mathrm{obj}(f)$ with $\mathrm{l_{Wf}}(o) = .\mathrm{Id}$* root *for the instance (extension) $f$. The set of roots for $f$ $\{r | r \in \mathrm{obj}(f)$ and $\mathrm{l_{Wf}}(r) = .\mathrm{Id}\}$ is denoted $\mathrm{Root}(f)$.*

This definition is well defined due to Lem. 4.12, because the definition of a range of a way-description relies on the definition of a folder of a class, which in turn relies on the derivation of class inclusion constraints.

We choose way-labels to be way descriptions over not only the schema $D$ but also the schema $D \cup \Upsilon$, and thus lay the foundation for satisfying the axioms and the derivable class inclusion constraints. Property 1 of extensions with labellings ensures that the class membership is correctly chosen, and property 2 ensures that way-labels are not arbitrarily chosen.

Example 4.29
*An instance with labelling for the schema $T$ and the set $\Omega$ of constraints in Exam. 4.10, and the set $\{h\}$ of classes is sketched in Fig. 4.4. The object $o_h$ is the only root of this instance.*

The prototype extension, an S-tree, for the counterexample of the completeness proof is an extension with labelling that contains only path functions as way-labels, because an S-tree merely aims at satisfying the axioms and the derivable class inclusion constraints. An S-tree is very similar to a *C-Tree* defined by Weddell [Wed89, Wed92], where $C$ stands in this case for a single class instead of a set of classes. Both kinds of trees possess adorned objects; but each C-Tree is an S-tree with labelling based on $\{C\}$ and the empty set of constraints.

The construction idea of an S-tree is to start with an object $r$, the *root* of the S-tree, to make the root an element of all classes derivable from a set $C$ of classes and the set $\Upsilon$ of constraints, and to introduce new objects as attribute values of the root $r$ due to the definedness axioms. These attribute values are made elements of classes according to the well-typedness axioms and the set $\Upsilon$ of constraints. We iterate this process of introducing attribute values for newly inserted objects. Instead of actually performing this process, we use path functions, precisely those that start at the classes in the set $C$.

To keep the information of why we insert an object, we adorn the object with the corresponding path function that leads to the insertion of this object. This adornment helps later on to prove properties of this object.

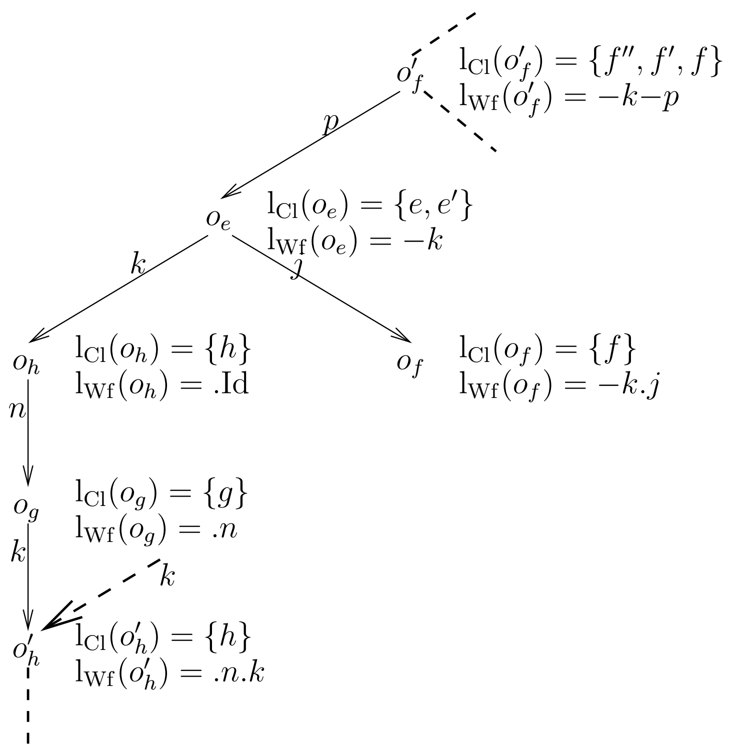

$o'_f$
$\mathrm{l_{Cl}}(o'_f) = \{f'', f', f\}$
$\mathrm{l_{Wf}}(o'_f) = -k-p$
$p$
$o_e$
$\mathrm{l_{Cl}}(o_e) = \{e, e'\}$
$\mathrm{l_{Wf}}(o_e) = -k$
$k$
$j$
$o_h$
$\mathrm{l_{Cl}}(o_h) = \{h\}$
$\mathrm{l_{Wf}}(o_h) = .\mathrm{Id}$
$o_f$
$\mathrm{l_{Cl}}(o_f) = \{f\}$
$\mathrm{l_{Wf}}(o_f) = -k.j$
$n$
$o_g$
$\mathrm{l_{Cl}}(o_g) = \{g\}$
$\mathrm{l_{Wf}}(o_g) = .n$
$k$
$k$
$o'_h$
$\mathrm{l_{Cl}}(o'_h) = \{h\}$
$\mathrm{l_{Wf}}(o'_h) = .n.k$

Figure 4.4: A sketch of the instance with labelling according to the parameters in Exam. 4.29

Definition 4.30 (S-Tree)
*Let $D$ be a schema with only the set* AX *as semantic constraints,* $\mathrm{SC}_D = \mathrm{AX}$, $\Upsilon$ *be a set of class inclusion constraints and onto constraints, and* $C \subset \mathrm{CLASS}_D$ *be a set of classes. An* S-tree *is an extension* $f := \langle \mathrm{pop}_f | \mathrm{ob}_f \rangle$ *of the schema $D$ with labelling based on the sets $C$ and $\Upsilon$ constructed as follows.*

**Step 1:** *For each* $p \in \bigcup_{c\in C} \mathrm{PathFuncs}_{D\cup\Upsilon}(c)$ *take a new constant $v$ and for all* $d \in \bigcup_{c\in C\cap\mathrm{Dom}_{D\cup\Upsilon}(p)} \mathrm{Ran}_{D\cup\Upsilon}(c,p)$ *add* $v : d$ *to* $\mathrm{pop}_f$*, and add an additional label* $\mathrm{l_{Wf}}(v)$ *assigned $p$.*

**Step 2:** *For each* $u, v \in \mathrm{obj}(f)$*, where* $p = \mathrm{l_{Wf}}(u)$ *and* $p \circ .m = \mathrm{l_{Wf}}(v)$*, add* $u[m \rightarrow v]$ *to* $\mathrm{ob}_f$*.*

Example 4.31
*An S-tree for the schema $T$ and the set $\Omega$ of constraints in Exam. 4.10, and the set $C = \{h\}$ of classes is outlined in Fig. 4.5.*

To show that an extension as defined above is an instance of the corresponding schema, we have to check that the completion of $f$ under $D$ satisfies the semantic constraints, which consist in this case only of unique name axioms, well-typedness axioms and definedness axioms. As we always take a new constant for objects and only one per path

$$\begin{tikzcd}
o_h \arrow[d, "n"'] & \begin{array}{l} \mathrm{l}_{\mathrm{Cl}}(o_h) = \{h\} \\ \mathrm{l}_{\mathrm{Wf}}(o_h) = .\mathrm{Id} \end{array} \\
o_g \arrow[d, "k"'] & \begin{array}{l} \mathrm{l}_{\mathrm{Cl}}(o_g) = \{g\} \\ \mathrm{l}_{\mathrm{Wf}}(o_g) = .n \end{array} \\
o'_h \arrow[d, dashed, no head] & \begin{array}{l} \mathrm{l}_{\mathrm{Cl}}(o'_h) = \{h\} \\ \mathrm{l}_{\mathrm{Wf}}(o'_h) = .n.k \end{array} \\
{} &
\end{tikzcd}$$

Figure 4.5: An outline of the S-tree according to Exam. 4.31

function, a violation of unique name axioms is not an issue here. The addition of attribute values is based on path functions as well. So no well-typedness and definedness axioms are violated. The satisfaction of the set $\Upsilon$ is not an issue at this point. The next lemma proves all as stated above.

LEMMA 4.32
*Let $f$ be an S-tree of a schema $D$ with only the set* AX *of axioms as semantic constraints,* $\mathrm{SC}_D = \mathrm{AX}$*, based on a set $C \subset \mathrm{CLASS}_D$ of classes and a set $\Upsilon$ of class inclusion constraints and onto constraints over the schema $D$. Then the S-tree $f$ is an instance of the schema $D$ with labelling based on the sets $C$ and $\Upsilon$.*

PROOF. To show that the S-tree $f$ is an instance of the schema $D$, we show that all remaining semantic constraints are satisfied, which are in this case only unique name axioms, definedness axioms and well-typedness axioms. But before the presentation of this proof, we show that properties 1 and 2 of instances with labellings are satisfied.

Property 1 is satisfied according to the first construction step of the S-tree $f$ and Lem. 4.12, because as no constraints apart from the axioms are present, the inference rules are complete in this case; and so we observe that for each object $o \in \mathrm{obj}(f)$, the set of classes this object $o$ is element of can be syntactically determined, which is done indeed due to the fact that the right side of the equation in property 1 relies on folders of classes.

$$\mathrm{l}_{\mathrm{Cl}}(o) = \bigcup_{c \in C \cap \mathrm{Dom}_{D\cup\Upsilon}(\mathrm{l}_{\mathrm{Wf}}(o))} \mathrm{Ran}_{D\cup\Upsilon}(c, \mathrm{l}_{\mathrm{Wf}}(o)) \ . \tag{4.6}$$

Property 2 is satisfied due to the second construction step of the S-tree $f$.

UNA: The sound and complete proof theory of F-logic reveals that there are three possible ways to introduce equality between F-logic objects:

1. by explicitly stating the equality,
2. by is-a acyclicity, and
3. by scalarity.

The first possibility can be ruled out because of the definitions of schemas and extensions, which simply lack this possibility. The second possibility cannot occur, because the schema $D$ has an acyclic class hierarchy. The third way cannot occur, because for each path function only one object is introduced.

DEF: Let $o \in \text{obj}(f)$ be an object according to the definition of S-tree $f$. This means for every attribute $m \in \text{Attr}_{D\cup\Upsilon}(c')$ for some class $c' \in \text{l}_{\text{Cl}}(o)$ exists a class $c \in C \cap \text{Dom}_{D\cup\Upsilon}(\text{l}_{\text{Wf}}(o))$ such that $c' \in \text{Ran}_{D\cup\Upsilon}(c, \text{l}_{\text{Wf}}(o))$. Therefore $\text{l}_{\text{Wf}}(o) \circ .m \in \text{PathFuncs}_{D\cup\Upsilon}(c)$ since $\text{l}_{\text{Wf}}(o)$ is a path function. According to the definition of S-tree $f$ there exists then an object $o'$ with $\text{l}_{\text{Wf}}(o') = \text{l}_{\text{Wf}}(o).m$, and hence $\text{ob}_f \models o[m \to o']$.

WT: Let $\text{ob}_f \models o[m \to o']$. Then $\text{l}_{\text{Wf}}(o) \circ .m = \text{l}_{\text{Wf}}(o')$.

1. Because the way-labels $\text{l}_{\text{Wf}}(o)$ and $\text{l}_{\text{Wf}}(o')$ are path functions, there exists a class $c' \in \text{Ran}_{D\cup\Upsilon}(c, \text{l}_{\text{Wf}}(o))$ for some class $c \in C \cap \text{Dom}_{D\cup\Upsilon}(\text{l}_{\text{Wf}}(o))$ with $\text{HIER}_{D\cup\Upsilon} \cup \text{SIG}_{D\cup\Upsilon} \models c'[m \Rightarrow ()]$, hence $\text{HIER}_D \cup \text{SIG}_D \models c'[m \Rightarrow ()]$. Because of (4.6) the class $c'$ is an element of $\text{l}_{\text{Cl}}(o)$, $c' \in \text{l}_{\text{Cl}}(o)$, and therefore $c'[m \Rightarrow ()]$ covers $o[m \to o']$.
2. Let $c \in \text{l}_{\text{Cl}}(o)$ be a class such that $\text{HIER}_D \cup \text{SIG}_D \models c[m \Rightarrow c']$ for some class $c'$. Due to (4.6), there exists $c'' \in C \cap \text{Dom}_{D\cup\Upsilon}(\text{l}_{\text{Wf}}(o))$ such that $c \in \text{Ran}_{D\cup\Upsilon}(c'', \text{l}_{\text{Wf}}(o))$. Finally, this means $c' \in \text{Ran}_{D\cup\Upsilon}(c'', \text{l}_{\text{Wf}}(o) \circ .m)$, and hence $c' \in \text{l}_{\text{Cl}}(o')$.

□

In general an S-tree does not satisfy onto constraints. So we modify an S-tree by adding a new object whenever an onto constraint is violated. This object refers to the object that gave rise to the violation via the corresponding attribute. But the addition of one object for every violation is not sufficient. The definedness axioms call for the addition of an S-tree for every object violating onto constraints. Then such an S-tree is pruned. The branch that starts with the attribute of the violated onto constraint is removed, and the object that led to the violation is grafted instead. The way-labels of the objects of the pruned-S-tree are prolonged to adjust the relative position of the objects to the new context.

Definition 4.33 (Pruned-S-Tree)
*Let $D$ be a schema with the set* AX *as semantic constraints,* $\text{SC}_D = \text{AX}$, $\Upsilon$ *be a set of class inclusion constraints and onto constraints over the schema $D$,* $C \subset \text{CLASS}_D$ *be a set of classes,* $m \in \bigcup_{c\in C} \text{Attr}_D(c)$ *be an attribute,* $w \in \text{W}_{\text{wf}}(D \cup \Upsilon)$ *be a way description such that* $w \circ -m$ *is a way description, and $o$ be a constant. A* pruned-S-tree *is an extension* $\text{pst}(C, o, m, w) := \langle \text{pop}|\text{ob} \rangle$ *of the schema $D$ with labelling based on the sets $C$ and $\Upsilon$ constructed as follows.*

**Step 1:** *For each* $p \in \bigcup_{c \in C} \mathrm{PathFuncs}_{D \cup \Upsilon}(c) \backslash \{x \in \mathrm{W}_{\mathrm{wf}}(D \cup \Upsilon) \mid ex.\ y \in \mathrm{W}_{\mathrm{arb}}(D \cup \Upsilon)\colon x \equiv .my\}$ *take a new constant* $v$ *and for all* $d \in \bigcup_{c \in C \cap \mathrm{Dom}_{D\cup\Upsilon}(p)} \mathrm{Ran}_{D \cup \Upsilon}(c, p)$ *add* $v : d$ *to* pop*, and add an additional label* $\mathrm{l}_{\mathrm{Wf}}(v)$ *assigned* $w \circ -m \circ p$*.*

**Step 2:** *For each* $u, v \in \mathrm{obj}(\mathrm{pst}(C, o, m, w))$*, where* $\mathrm{l}_{\mathrm{Wf}}(u) \circ .n = \mathrm{l}_{\mathrm{Wf}}(v)$*, add* $u[n \to v]$ *to* ob*.*

**Step 3:** *Add* $o'[m \to o]$ *to* ob*, where* $o'$ *is the object with way labelling* $w \circ -m$*,* $\mathrm{l}_{\mathrm{Wf}}(o') = w \circ -m$*, i. e.,* $o'$ *is the object added for the path function* .Id*. Because of this "historical" background we call the object* $o'$ *a* root *as well.*

Technically, $pst(\cdots)$ is not an extension. The addition of the data atom $o'[m \to o]$ in the last step thwarts that, because the object $o$ is not a member of any class. We neglect this minor defect, because pruned-S-trees are not a "stand-alone" concept.

Example 4.34
*When we look at the S-tree in Exam. 4.31, which was constructed for the schema* $T$*, the set* $\Omega$ *of constraints in Exam. 4.10 and the set* $C = \{h\}$ *of classes, we notice that the object* $o_h$ *violates the onto constraint* $e\{k|h\}$*. We fix this violation by inserting the object* $o_e$ *and by making the object* $o_h$ *the value for the attribute* $k$ *of the object* $o_e$*. The object* $o_e$ *is a member of the classes* $e$ *and* $e'$*. So the definedness axioms require that a value for the attribute* $j$ *of this object is defined, because the attribute* $j$ *is declared on the class* $e$*. Therefore we insert not only the object* $o_e$ *but instead the pruned-S-tree* $\mathrm{pst}(\{e, e'\}, o_h, k, .\mathrm{Id})$ *shown in Fig. 4.6. The parameter* .Id *passed for the construction indicates where the root of the pruned-S-tree is inserted relatively to the root of the S-tree, while the argument* $o_h$ *indicates the absolute position of the pruned-S-tree.*

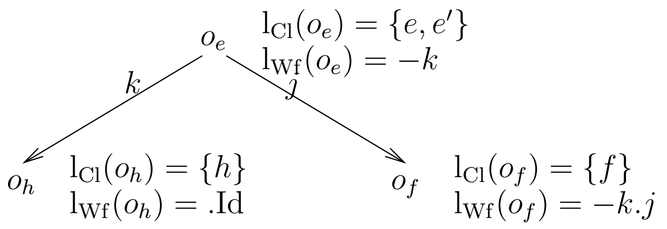

$o_e$
$\mathrm{l}_{\mathrm{Cl}}(o_e) = \{e, e'\}$
$\mathrm{l}_{\mathrm{Wf}}(o_e) = -k$
$k$
$j$
$o_h$
$\mathrm{l}_{\mathrm{Cl}}(o_h) = \{h\}$
$\mathrm{l}_{\mathrm{Wf}}(o_h) = .\mathrm{Id}$
$o_f$
$\mathrm{l}_{\mathrm{Cl}}(o_f) = \{f\}$
$\mathrm{l}_{\mathrm{Wf}}(o_f) = -k.j$

Figure 4.6: The pruned-S-tree $\mathrm{pst}(\{e, e'\}, o_h, k, .\mathrm{Id})$ from Exam. 4.34

We determine the set of objects that violate onto constraints. Each of these objects should then be referred to by a root of a pruned-S-tree. It might be the case that an object violates several onto constraints having the same attribute. In this case only one pruned-S-tree is introduced.

Definition 4.35 (Unsat)
*Let* $D$ *be a schema with only the set* AX *of axioms as semantic constraints,* $\mathrm{SC}_D = \mathrm{AX}$*,* $\Upsilon$ *be a set of class inclusion constraints and onto constraints over the schema* $D$*,* $C \subset \mathrm{CLASS}_D$ *be a set of classes, and* $f$ *be an instance of the schema* $D$ *with labelling based*

*on the sets $C$ and $\Upsilon$. The set of* unsatisfied objects under $\Upsilon$ with the violated classes and attributes *is*

$$\begin{aligned}\operatorname{unsat}(f,\Upsilon) := \{(\{d \mid \textit{ex. } c\colon c \in \mathrm{l}_{\mathrm{Cl}}(o) \textit{ and } d\{m|c\} \in \Upsilon^{+_D} \textit{ and}\\ \textit{for all } o' \in \operatorname{obj}(f)\colon d \notin \mathrm{l}_{\mathrm{Cl}}(o') \textit{ or } \mathrm{ob}_f \not\models o'[m \to o]\},\\ o, m) \mid o \in \operatorname{obj}(f) \textit{ and } m \in \mathrm{Attr}_D\}\ .\end{aligned}$$

Example 4.36
*The set* $\operatorname{unsat}(f,\Omega)$ *for the S-tree $f$ in Exam. 4.31 and the set $\Omega$ of constraints in Exam. 4.10 is*

$$\operatorname{unsat}(f,\Omega) = \{(\{e,e'\}, o_h, k), (\emptyset, o_h, n), \ldots\}\ .$$

*In Exam. 4.34 we utilised the information of the vector* $(\{e,e'\}, o_h, k)$ *for the parameters in the construction of the pruned-S-tree* $\operatorname{pst}(\{e,e'\}, o_h, k, .\mathrm{Id})$ *where the way description* $.\mathrm{Id}$ *is the way-label of the object* $o_h$, $\mathrm{l}_{\mathrm{Wf}}(o_h) = .\mathrm{Id}$.

In the next construction step we want to ensure that violated onto constraints are satisfied. Therefore, for every object that violates an onto constraint, we introduce a new object referring to that particular object. But instead of creating just one object, we insert a pruned-S-tree that has as root the inserted object.

Definition 4.37 (Extrev)
*Let $D$ be a schema with only the set* AX *as semantic constraints,* $\mathrm{SC}_D = \mathrm{AX}$, *$\Upsilon$ be a set of class inclusion constraints and onto constraints over the schema $D$, $C \subset \mathrm{CLASS}_D$ be a set of classes, and $f$ be an instance of the schema $D$ with labelling based on the sets $C$ and $\Upsilon$. Then the extension* $\operatorname{extrev}_\Upsilon(f)$ *of the schema $D$ with labelling based on the sets $C$ and $\Upsilon$ is defined as*

$$\operatorname{extrev}_\Upsilon(f) := f \cup \bigcup_{\substack{(C,o,m) \in \operatorname{unsat}(f,\Upsilon) \textit{ and}\\ C \neq \emptyset}} \operatorname{pst}(C, o, m, \mathrm{l}_{\mathrm{Wf}}(o))\ ,$$

*where every object introduced in one of the pruned-S-trees does not occur in neither any other pruned-S-tree nor the instance $f$.*

Both the definition of unsat and the definition of extrev take instances of the underlying schema with labelling as input, because only the initial extension is an S-tree. The output of the operation extrev is then further used as input again. To enable this iteration, the output has to be an instance with labelling again as we show in Lem. 4.39.

Example 4.38
*Coming back to the S-tree $f$ in Exam. 4.31, we apply the operation* extrev *on this S-tree with the set $\Omega$ of constraints in Exam. 4.10. Since the vector* $(\{e,e'\}, o_h, k)$ *is an element of the set* $\operatorname{unsat}(f,\Omega)$, *we add among others the pruned-S-tree* $\operatorname{pst}(\{e,e'\}, o_h, k, \mathrm{l}_{\mathrm{Wf}}(o_h))$ *as depicted in Fig. 4.6 and get the extension with labelling as shown in Fig. 4.7 as result.*

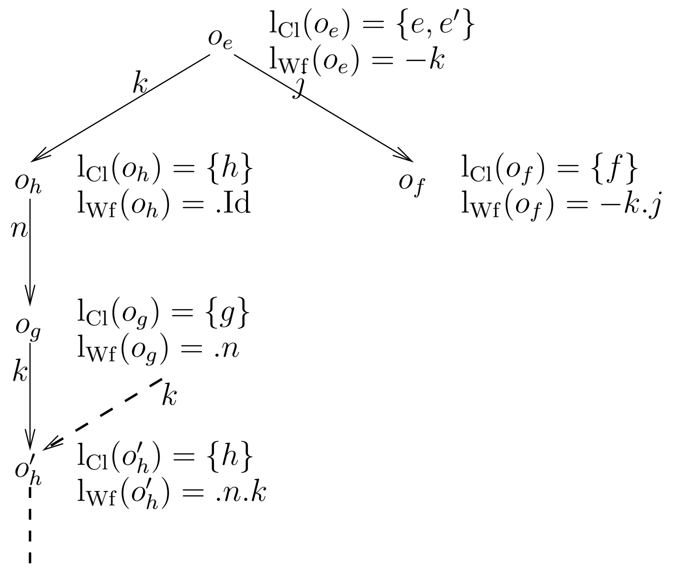

$o_e$
$\mathrm{l}_{\mathrm{Cl}}(o_e) = \{e, e'\}$
$\mathrm{l}_{\mathrm{Wf}}(o_e) = -k$
$k$
$j$
$o_h$
$\mathrm{l}_{\mathrm{Cl}}(o_h) = \{h\}$
$\mathrm{l}_{\mathrm{Wf}}(o_h) = .\mathrm{Id}$
$o_f$
$\mathrm{l}_{\mathrm{Cl}}(o_f) = \{f\}$
$\mathrm{l}_{\mathrm{Wf}}(o_f) = -k.j$
$n$
$o_g$
$\mathrm{l}_{\mathrm{Cl}}(o_g) = \{g\}$
$\mathrm{l}_{\mathrm{Wf}}(o_g) = .n$
$k$
$k$
$o'_h$
$\mathrm{l}_{\mathrm{Cl}}(o'_h) = \{h\}$
$\mathrm{l}_{\mathrm{Wf}}(o'_h) = .n.k$

Figure 4.7: Operation extrev applied on the S-tree in Fig. 4.5

LEMMA 4.39
*Let $D$ be a schema with only the set* AX *as semantic constraints,* $\mathrm{SC}_D = \mathrm{AX}$*,* $\Upsilon$ *be a set of class inclusion constraints and onto constraints over the schema $D$, $C \subset \mathrm{CLASS}_D$ be a set of classes, and $f$ be an instance of the schema $D$ with labelling based on the sets $C$ and $\Upsilon$. Then* $\mathrm{extrev}_\Upsilon(f)$ *is an instance of the schema $D$ with labelling based on the sets $C$ and $\Upsilon$.*

PROOF. That every pruned-S-tree is nearly an instance can be proven along the same lines of arguments as in the proof of Lem. 4.32 except for the satisfaction of the well-typedness axioms and the definedness axioms for the pruned attribute. But then the definedness axioms are satisfied, because the pruned attribute receives the object violating onto constraints as value. The satisfaction of the well-typedness axioms for the pruned attribute remains to be shown.

So we consider the root $o'$ of a pruned-S-tree introduced for the set of classes $E$, object $o$ and attribute $m$, and $a \in \mathrm{l}_{\mathrm{Cl}}(o')$ be a class with

$$\mathrm{HIER}_D \cup \mathrm{SIG}_D \models a[m \Rightarrow b] \ . \tag{4.7}$$

According to the definition of a pruned-S-tree and Lem. 4.12

$$\mathrm{l}_{\mathrm{Cl}}(o') \overset{\text{Def.4.28}}{=} \bigcup_{e \in E \cap \mathrm{Dom}_{D\cup\Upsilon}(.\mathrm{Id})} \mathrm{Ran}_{D\cup\Upsilon}(e, .\mathrm{Id}) \overset{\text{Def.4.23}}{=} \bigcup_{e \in E} \mathrm{Folder}_{D\cup\Upsilon}(e) \ .$$

Therefore, there exists a class $a' \in E$ with $a' \subset a$, $a'\{m|b'\} \in \Upsilon^{+_D}$ and $b' \in \mathrm{l}_{\mathrm{Cl}}(o)$. By inference rule domain relaxation, we have

$$a\{m|b'\} \in \Upsilon^{+_D} \ . \tag{4.8}$$

Thus $b' \subset b$ follows from signature inclusion with (4.7) and (4.8). Consequently, we have $b \in \mathrm{l}_{\mathrm{Cl}}(o)$ and hence the satisfaction of all well-typedness axioms.

Conditions 1 and 2 of Def. 4.28 are also satisfied, because every pruned-S-tree is nearly an instance with labelling except for the root of the pruned-S-tree and the pruned attribute. But the root of the pruned-S-tree satisfies condition 1 by definition as well as condition 2. □

Having the instance with labelling that violates the onto constraints in a set of constraints, we can fix the violations by applying the operation extrev to that instance; but we purchase thereby new violations of onto constraints, which we fix by iterating the application of the operation. The final outcome is then an instance of the underlying schema $D$ with labelling that satisfies the set of constraints as well.

Lemma 4.40
*Let $D$ be a schema with only the set* AX *of axioms as set of semantic constraints,* $\mathrm{SC}_D = \mathrm{AX}$*,* $\Upsilon$ *be a set of class inclusion constraints and onto constraints over the schema $D$, $C \subset \mathrm{CLASS}_D$ be a set of classes, and $f$ be an instance of the schema $D$ with labelling based on the sets $C$ and $\Upsilon$. Then $\bigcup_{i\in\mathbb{N}} \mathrm{extrev}^i_\Upsilon(f)$ is an instance of the schema $D \cup \Upsilon$ with labelling based on the sets $C$ and $\Upsilon$.*

Proof. Because of Lem. 4.39, $\mathrm{extrev}^i_\Upsilon(f)$ is an instance of the schema $D$ with labelling based on the sets $C$ and $\Upsilon$ provided $f$ is an instance of the schema $D$ with labelling based on the sets $C$ and $\Upsilon$. Therefore $\bigcup_{i\in\mathbb{N}} \mathrm{extrev}^i_\Upsilon(f)$ is an instance of the schema $D$ with labelling based on the sets $C$ and $\Upsilon$. It remains to show that the extension $\bigcup_{i\in\mathbb{N}} \mathrm{extrev}^i_\Upsilon(f)$ satisfies the set $\Upsilon$ of class inclusion constraints and onto constraints.

Due to Def. 4.30 and Lem. 4.12 each derivable class inclusion constraint is satisfied in any extension $\mathrm{extrev}^i_\Upsilon(f)$ and hence in $\bigcup_{i\in\mathbb{N}} \mathrm{extrev}^i_\Upsilon(f)$.

If object $o$ is generated in the $i$-th iteration, then all onto constraints that are violated by $o$ are satisfied in the $i+1$-th iteration. This continues to be so throughout further iterations. □

Finally we are ready to prove the completeness of the inference rules for class inclusion constraints and onto constraints. The proof will be conducted by contraposition. We assume that we cannot derive a constraint $\upsilon$ from the set $\Upsilon$ of constraints, $\Upsilon \not\vdash_D \upsilon$. Then we construct an instance $f$ of the schema $D$ such that the extension $f$ is

- an instance of the schema $D \cup \Upsilon$,
- but not an instance of the schema $D \cup \{\upsilon\}$.

Theorem 4.41
*Let $\Upsilon \cup \{\upsilon\}$ be a set of class inclusion constraints and onto constraints over a schema $D$ with only the set* AX *of axioms as semantic constraints,* $\mathrm{SC}_D = \mathrm{AX}$*, such that the constraint $\upsilon$ cannot be derived from the set $\Upsilon$, $\Upsilon \not\vdash_D \upsilon$. Then there exists an instance $f$ of the schema $D$ such that the extension $f$ is*

- *an instance of the schema $D \cup \Upsilon$,*
- *but not an instance of the schema $D \cup \{v\}$.*

Proof. Let the constraint $v$ be of the form $c \subset d$ or $d\{m|c\}$. Then let $g$ be an S-tree built for $D$, the sets $\{c\}$ and $\Upsilon$, and the object $r$ be its root, $\text{Root}(g) = \{r\}$. Due to Lem. 4.32 the S-tree $g$ is an instance of the schema $D$ with labelling, and therefore, by Lem. 4.40, $f := \bigcup_{i \in \mathbb{N}} \text{extrev}^i_\Upsilon(g)$ is an instance of the schema $D \cup \Upsilon$ with labelling based on the sets $\{c\}$ and $\Upsilon$. It remains to show that the extension $f$ does not satisfy the constraint $v$.

We prove by contradiction that the extension $f$ does not satisfy the constraint $v$, be it a class inclusion constraint or an onto constraint. So we assume conversely that the extension $f$ satisfies the constraint $v$.

If the constraint $v$ takes on the form $c \subset d$, then the root $r$ is a member of the class $d$, $d \in \text{l}_{\text{Cl}}(r)$. Due to Def. 4.28, the fact that the S-tree is constructed only for the class $c$ and the fact that the object $r$ is a root, $\text{l}_{\text{Wf}}(r) = .\text{Id}$, the equation

$$\text{l}_{\text{Cl}}(r) = \text{Folder}_D(c) \tag{4.9}$$

holds. So in contradiction to the assumption we derive $c \subset d$, because $d \in \text{l}_{\text{Cl}}(r) = \text{Folder}_D(c)$.

If the constraint $v$ takes on the form $d\{m|c\}$, then there exists an object $o$ with $\text{ob}_f \models o[m \to r]$. Because of the construction of the S-tree $g$, the S-tree $g$ does not satisfy the onto constraint $d\{m|c\}$. So the object $o$ must have been introduced in the first application of the operation extrev on the S-tree $g$. This means there exists a vector $(C, r, m) \in \text{unsat}(g, \Upsilon)$ with $C \neq \emptyset$. Consequently, there is a class $c' \in \text{l}_{\text{Cl}}(r)$ and an onto constraint $d'\{m|c'\} \in \Upsilon^{+_D}$ such that $d \in \text{Folder}_D(d')$, hence $d' \subset d$ and $d\{m|c'\} \in \Upsilon^{+_D}$, because the onto constraint can only be satisfied if $m$ is an attribute declared for the class $d$, $m \in \text{Attr}_D(d)$. Finally, (4.9) holds in this case as well and (4.9) implies $c \subset c'$, and therefore we derive the onto constraint $d\{m|c\}$ from the set $\Upsilon$, $d\{m|c\} \in \Upsilon^{+_D}$, which is a contradiction.

□

Thus we have proven the completeness of the inference rules for class inclusion constraints and onto constraints.

## 4.4 Path Functional Dependencies

Path functional dependencies as introduced by Weddell [Wed89, Wed92] are an extension of functional dependencies as known in the relational data model. While functional dependencies are defined on a relation scheme, path functional dependencies are declared for classes and employ path functions in lieu of simple attributes in there left-hand and right-hand sides. A path functional dependency then states if two objects in the class given in the dependency deliver identical results for each path function in the left side of the dependency, the objects have to agree on the value for each path function in the right side.

Example 4.42
*Again, we come back to Exam. 3.2 and extend it by semantic constraints that reflect restrictions imposed on instances of the class* Phone-admin. *They require that if two paths of the form stated in the left-hand side deliver identical results, so do the corresponding paths of the form stated in the right-hand side. In this example we have only path functions of length* 1 *in the path functional dependencies. So in fact they are purely functional dependencies. These path functional dependencies are the starting point to perform pivoting in Exam. 7.3.*

*The first path functional dependency demands that for every* Phone-admin *object the value for attribute* fac *uniquely determines the value for attribute* sch. *So any objects of class* Phone-admin *agree on their value for the attribute* sch *whenever they agree on the value for attribute* fac.

$$\begin{array}{l}\text{Phone-admin}(.\text{fac} \rightarrow .\text{sch})\\ \text{Phone-admin}(.\text{fac} \rightarrow .\text{ph})\\ \text{Phone-admin}(.\text{sch} \rightarrow .\text{dep})\\ \text{Phone-admin}(.\text{ph} \rightarrow .\text{dep})\end{array}$$

*In general, both sides of a path functional dependency can also contain sequences of path functions where the pertinent conditions on each path function are understood to be conjunctively connected.*

*Examples 7.2 and 7.3 show how path functional dependencies with longer path functions look like.*

The path functions in a path functional dependency are chosen in a way that they start all at the class the path functional dependency is declared for. In that way, we guarantee when we apply one of these path functions to an object of the respective class, we always obtain a defined result.

We have to bear in mind here that the set of path functions starting at a class is influenced by class inclusion constraints and onto constraints. When we derive a class inclusion constraints $c \subset d$, every path function or better way description starting at the class $d$ starts also at the class $c$.

Definition 4.43 (Path Functional Dependency)
*Let $D$ be a database schema, and $c \in \text{CLASS}_D$ be a class.*

**Syntax:** *A* path functional dependency *for the class $c$ is of the form*

$$c(p_1 \cdots p_k \rightarrow p_{k+1} \cdots p_n) \ ,$$

*where $0 < k < n$, and where $p_i \in \text{PathFuncs}_D(c)$ for $i \in \{1, \ldots, n\}$.*

**Semantics:** *To define that a database instance satisfies a path functional dependency, we construct F-formulae. We facilitate this task by using a term $X[p \rightarrow Y]$ with*

$$p \equiv .m_1^p \cdots .m_l^p \in \text{P}_{\text{arb}}(D) \backslash \{.\text{Id}\}$$

*as a shorthand form for*

$$(\exists X_1^p \cdots \exists X_{l-1}^p (X[m_1^p \rightarrow X_1^p] \wedge X_1^p[m_2^p \rightarrow X_2^p] \wedge \cdots \wedge X_{l-1}^p[m_l^p \rightarrow Y])) \ ,$$

*and the term* $X[.\mathrm{Id} \rightarrow Y]$ *as a shorthand form for* $X \overset{\circ}{=} Y$.

*For a path functional dependency*

$$c(p_1 \cdots p_k \rightarrow p_{k+1} \cdots p_n)$$

*the path functional dependency F-formulae are*

$$\begin{aligned} X[p \rightarrow P] \longleftarrow\ & X : c[p_1 \rightarrow P_1; \ldots; p_k \rightarrow P_k] \wedge \\ & Y : c[p_1 \rightarrow P_1; \ldots; p_k \rightarrow P_k; p \rightarrow P] \end{aligned}$$

*for each* $p \in \{p_{k+1}, \ldots, p_n\}$.

*The set of path functional dependencies in* $\mathrm{SC}_D$ *is denoted by* $\mathrm{PFD}_D$.

The F-formulae in the preceding definition are no longer F-Horn-rules, as they contain existentially bound variables, but we assume that all visible variables are universally bound. They are instead in the so-called *implicative normal form.* Although formally the F-formulae above are no F-Horn-rules, we make the following observation. An existentially bound variable in a formula

$$(\exists X_1^p \cdots \exists X_{l-1}^p (X[m_1^p \rightarrow X_1^p] \wedge X_1^p[m_2^p \rightarrow X_2^p] \wedge \cdots \wedge X_{l-1}^p[m_l^p \rightarrow Y]))$$

depends in a sense only on the variable $X$ and its assigned values, because the methods are scalar methods and the presence of definedness axioms in our schemas.

From now onwards, we limit the set of semantic constraints in a schema to class inclusion constraints, onto constraints and path functional dependencies plus unique name axioms, well-typedness axioms and definedness axioms, i. e., a schema consists of the following components unless otherwise said:

$$D = \langle \mathrm{CLASS}_D | \mathrm{METH}_D | \mathrm{HIER}_D | \mathrm{SIG}_D | \mathrm{AX} \cup \mathrm{CIC}_D \cup \mathrm{OC}_D \cup \mathrm{PFD}_D \rangle \ .$$

We simply call a set of class inclusion constraints, onto constraints and path functional dependencies a *set of constraints* in the remainder of this work.

## 4.5 Inference Rules for CICs, OCs and PFDs

Again as in the case for class inclusion constraints and onto constraints, we need inference rules for path functional dependencies to make the decision problem of the implication amenable to an algorithmic treatment.

To be more precise, we need inference rules for class inclusion constraints, onto constraints and path functional dependencies. It turns out that class inclusion constraints and onto constraints affect the implication of path functional dependencies, but not the other way round. The impact of class inclusion constraints and onto constraints already starts at the declaration of path functional dependencies, because these constraints potentially enlarge the set of path functions for a class. Because of this one-way influence, a fact that we prove later in Theor. 4.55, the inference rules C1 to C5 are even complete when path functional dependencies are added to the set of semantic constraints. Then we supplement these inference rules to capture the implication of class inclusion constraints, onto constraints and path functional dependencies.

Weddell [Wed89, Wed92] presents five inference rules for path functional dependencies alone, which we adapt to our data model as well. The first rule (*A1: reflexivity*) captures the reflexivity of path functional dependencies. The second rule (*A2: path function augmentation*) allows to extend both sides of a path functional dependency by a set of path functions. The third rule (*A3: transitivity*) shows that path functional dependencies are transitive. These three rules exist in similar forms, with only attributes instead of path functions, for relational functional dependencies.

The next two rules *simple attribution* and *simple prefix augmentation* have no counterparts in the relational data model. The rule *simple attribution* says that an object uniquely determines its attribute values. While the first four rules act locally on one class only, *simple prefix augmentation* includes two classes. If a path functional dependency can be derived for a class $c_2$ and this class is reachable from a class $c_1$ via the attribute $a$, then the path functional dependency that results from prolonging each path function in the original path functional dependency declared for the class $c_2$ can be derived for the class $c_1$.

The following inference rule (*A6: simple prefix reduction*) is the inverse of simple prefix augmentation. Simple prefix reduction is described by Thalheim [Tha93a] under the name *reduction*. Simple prefix reduction allows to telescope the path functions in a derived path functional dependency provided the path functions have a common prefix and an onto constraint is derivable with the common prefix as attribute. The last rule (*A7: path functional dependency inheritance*) brings class inclusion constraints into play. A path functional dependency declared for a class $c_2$ holds also for every subclass of the class $c_2$ where we use the term subclass to denote both real subclasses and those classes for which we can derive a corresponding class inclusion constraint. So the word inheritance in the name of the rule is actually a misnomer.

For the inference rules C1 to C5 we use the notation $\vdash_D$ to denote a derivation of a class inclusion constraint or onto constraint. Since the inference rules A1 to A7 produce only path functional dependency and therefore no conflicts arise, we reuse this notation even for these rules.

Definition 4.44 (Inference Rules)
*Let $\Xi$ be the set of constraints in a schema $D$, $\Upsilon$ be a set of class inclusion constraints and onto constraints in the schema $D$, and $\pi$ be a path functional dependency over the schema $D$. $\pi$ is* derivable *from the set $\Xi$, written $\Xi \vdash_D \pi$,* iff *it is a member of the set $\Xi$ or is the result of one or more applications of the following inference rules.*

**A1.** Reflexivity: *For every class $c \in \mathrm{CLASS}_D$ and for all non-empty $Y \subset X$ where $X$*

*is a finite subset of* PathFuncs$_D(c)$, *derive* $c(X \rightarrow Y)$. *Abbreviated*

$$\frac{\emptyset}{c(X \rightarrow Y)}{}^{A1} \ .$$

**A2.** Path function augmentation: *For every class* $c \in \text{CLASS}_D$ *and finite subsets* $X$, $Y$ *and* $Z$ *of* PathFuncs$_D(c)$, *if* $c(X \rightarrow Y)$ *can be derived, then so can* $c(XZ \rightarrow YZ)$ *(where* $XZ$*, for example, denotes the union of all path functions in* $X$ *and* $Z$*). Abbreviated*

$$\frac{\{c(X \rightarrow Y)\}}{c(XZ \rightarrow YZ)}{}^{A2} \ .$$

**A3.** Transitivity: *For every class* $c \in \text{CLASS}_D$ *and finite subsets* $X, Y, Z \subset$ PathFuncs$_D(c)$, *if both* $c(X \rightarrow Y)$ *and* $c(Y \rightarrow Z)$ *can be derived, then so can* $c(X \rightarrow Z)$. *Abbreviated*

$$\frac{\{c(X \rightarrow Y), c(Y \rightarrow Z)\}}{c(X \rightarrow Z)}{}^{A3} \ .$$

**A4.** Simple attribution: *For every class* $c \in \text{CLASS}_D$ *and attribute* $a \in \text{Attr}_D(c)$, *derive* $c(.\text{Id} \rightarrow .a)$. *Abbreviated*

$$\frac{\emptyset}{c(.\text{Id} \rightarrow .a)}{}^{A4} \ .$$

**A5.** Simple prefix augmentation: *For every class* $c_1 \in \text{CLASS}_D$ *and attribute* $a \in \text{Attr}_D(c_1)$, *if*

$$c_2(p_1 \cdots p_m \rightarrow p_{m+1} \cdots p_n)$$

*can be derived, where* $c_2 \in \text{Ran}_D(c_1, .a)$, *then so can*

$$c_1(.a \circ p_1 \cdots .a \circ p_m \rightarrow .a \circ p_{m+1} \cdots .a \circ p_n) \ .$$

*Abbreviated*

$$\frac{\{c_2(p_1 \cdots p_m \rightarrow p_{m+1} \cdots p_n)\}}{c_1(.a \circ p_1 \cdots .a \circ p_m \rightarrow .a \circ p_{m+1} \cdots .a \circ p_n)}{}^{A5} \ .$$

**A6.** Simple prefix reduction: *For every class* $c_1 \in \text{CLASS}_D$, *if*

$$c_2(.a \circ p_1 \cdots .a \circ p_m \rightarrow .a \circ p_{m+1} \cdots .a \circ p_n)$$

*can be derived, where* $c_2\{a|c_1\} \in \Upsilon^{+_D}$, *then so can*

$$c_1(p_1 \cdots p_m \rightarrow p_{m+1} \cdots p_n) \ .$$

*Abbreviated*

$$\frac{\{c_2(.a \circ p_1 \cdots .a \circ p_m \rightarrow .a \circ p_{m+1} \cdots .a \circ p_n), c_2\{a|c_1\}\}}{c_1(p_1 \cdots p_m \rightarrow p_{m+1} \cdots p_n)}{}^{A6} \ .$$

**A7.** Path functional dependency inheritance: *For every class $c_1 \in \mathrm{CLASS}_D$, if $c_2(X \to Y)$ can be derived where $c_1 \subset c_2 \in \Upsilon^{+_D}$, then so can $c_1(X \to Y)$. Abbreviated*

$$\frac{\{c_2(X \to Y)\}}{c_1(X \to Y)}{\scriptstyle A7} \ .$$

In the remainder of this section we show that the inference rules are sound and complete. As in Sections 4.1, 4.2 and 4.3 we present a running example to illustrate the concepts used in these proofs, but again there are too many aspects to be captured for presenting an example that offers an interpretation in the real world. To compensate this lack, we give an example of how some of the inference rules do their job for a variant of the schema in Exam. 3.2.

Example 4.45

*In example 7.3 we will present a schema that is result of a transformation on the schema in Exam. 3.2. This schema, which is depicted in Fig. 4.8, includes path functional dependencies and onto constraints. We assume that at least the semantic constraints*

$$\textsf{Phone-admin}(\textsf{.fac} \to \textsf{.fac.sch})$$

*and*

$$\textsf{Phone-admin}\{\textsf{fac}|\textsf{Phoning-faculty}\}$$

*are part of this schema. We can derive the path functional dependency*

$$\textsf{Phoning-faculty}(.\mathrm{Id} \to \textsf{.sch})$$

*from these semantic constraints by "simple prefix reduction". We can even show that*

$$\textsf{Phone-admin}(\textsf{.fac} \to \textsf{.fac.sch})$$

*is redundant. By "simple prefix augmentation", we can derive from the trivial dependency*

$$\textsf{Phoning-faculty}(.\mathrm{Id} \to \textsf{.sch})$$

*(by "simple attribution") the dependency*

$$\textsf{Phone-admin}(\textsf{.fac} \to \textsf{.fac.sch}) \ .$$

Example 4.46

*For the demonstration of how the inference rules work, we declare a schema $S$ with the components as depicted in Fig. 4.9. The schema is graphically depicted in Fig. 4.10.*

*Noteworthy is in this example the path functional dependency $f''(.p.j \to .q)$, because the path function $.q$ in the right side is only a path function starting at the class $f''$ due to the fact that the class inclusion constraints $f'' \subset f$ holds. In this derivation we employ the class inclusion constraint $f' \subset f$, which we derive by signature inclusion with $e\{j|f'\}$ and $e[j \Rightarrow f]$. From the set $\mathrm{CIC}_S \cup \mathrm{OC}_S \cup \mathrm{PFD}_S$ we derive the path functional dependencies as depicted in Fig. 4.11.*

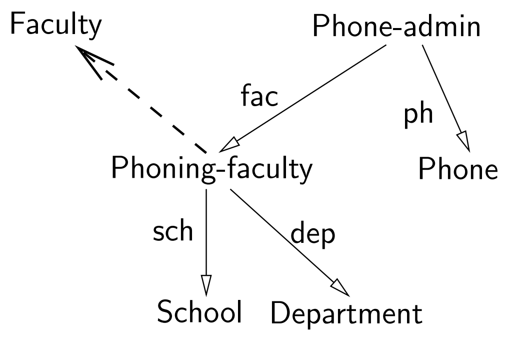


Figure 4.8: An F-logic schema

$$
\begin{aligned}
\mathrm{CLASS}_S &= \{d, e, e', f, f', f'', g, h, i\}\ ,\\
\mathrm{METH}_S &= \{j, k, n, p, q, u, v, w\}\ ,\\
\mathrm{HIER}_S &= \{e{::}e', f''{::}f'\}\ ,\\
\mathrm{SIG}_S &= \{e[j \Rightarrow f], e'[k \Rightarrow h],\\
&\qquad f[q \Rightarrow i; w \Rightarrow i],\\
&\qquad f''[p \Rightarrow e],\\
&\qquad h[n \Rightarrow g; v \Rightarrow d],\\
&\qquad g[u \Rightarrow h]\}\ ,\\
\mathrm{SC}_S &= \mathrm{AX}\\
&\quad \cup\\
&\quad \big\{e\{j|f'\}, e\{k|h\}, f''\{p|e\}\big\}\\
&\quad \cup\\
&\quad \big\{f(.q \rightarrow .w), f''(.p.k.v \rightarrow .p.j), f''(.p.j \rightarrow .q), h(.v \rightarrow .n.u)\big\}
\end{aligned}
$$

Figure 4.9: The schema components for the Exam. 4.46

In the following sections we need to know which constraints, be it class inclusion constraints, onto constraints or path functional dependencies, we can derive from a set of constraints in a schema by means of the inference rules C1 to C5 and the inference rules A1 to A7. In addition, we are sometimes only interested in the path functional dependencies that we can derive from a set of constraints in a schema.

In the relational data model we can determine the set of attributes over a relational scheme that are functionally determined by another set of attributes over this scheme. Likewise we can determine the set of path functions starting at a class that is functionally determined by another set of path functions starting at the same class.

Definition 4.47 (Closure)
*Let $\Xi$ be a set of constraints over a schema $D$.*

1. *The* closure *of the set $\Xi$, written $\Xi^{+_D}$, is the set of all constraints $\xi$ over the schema*

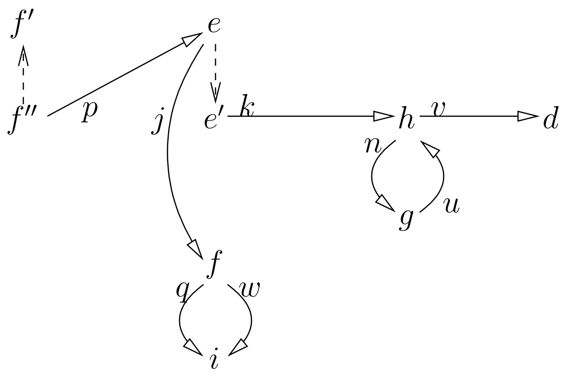


Figure 4.10: The schema for the Exam. 4.46

$$
\begin{array}{rl}
 & \overset{\text{A1}}{\vdash_S} h(\{.v,.n\} \to .v) \\
h(.v \to .n.u) & \overset{\text{A2}}{\vdash_S} h(\{.v,.n\} \to \{.n.u,.n\}) \\
\left.\begin{array}{l} h(\{.v,.n\} \to .v) \\ h(.v \to .n.u) \end{array}\right\} & \overset{\text{A3}}{\vdash_S} h(\{.v,.n\} \to .n.u) \\
 & \overset{\text{A4}}{\vdash_S} h(.\mathrm{Id} \to .n) \\
h(.v \to .n.u) & \overset{\text{A5}}{\vdash_S} e'(.k.v \to .k.n.u) \\
\left.\begin{array}{l} f''(.p.k.v \to .p.j) \\ f''\{p|e\} \end{array}\right\} & \overset{\text{A6}}{\vdash_S} e(.k.v \to .j) \\
\left.\begin{array}{l} f(.q \to .w) \\ f' \subset f \end{array}\right\} & \overset{\text{A7}}{\vdash_S} f'(.q \to .w)
\end{array}
$$

Figure 4.11: Examples for the application of A1 to A7

*$D$ where $\Xi \vdash_D \xi$.*

2. *The* PFD-closure *of the set $\Xi$, written $\Xi^{\oplus_D}$, is the set of all path functional dependencies $\pi$ over the schema $D$ where $\Xi \vdash_D \pi$.*

3. *The* closure of a finite, non-empty set of path functions $X \subset \mathrm{PathFuncs}_D(c)$ *for some class $c \in \mathrm{CLASS}_D$, written $X^{+_{D,c}}$, is the set of all path functions $p \in \mathrm{PathFuncs}_D(c)$ where $\Xi \vdash_D c(X \to p)$.*

As for the inference rules C1 to C5, showing the correctness of the inference rules A1 to A7 is the easier part.

Theorem 4.48 (Soundness of the Inference Rules A1 to A7)
*The inference rules A1 to A7 are* sound, *i. e., for the set $\Xi$ of constraints in a schema $D$ and a path functional dependency $\pi$ over the schema $D$: if $\pi \in \Xi^{+_D}$, then every instance of the schema $D$ satisfies the path functional dependency $\pi$,* $\operatorname{sat}(D) \subset \operatorname{sat}(D \cup \{\pi\})$.

Proof. Showing the correctness of the inference rules is straightforward. It is possible to use theorem proving for this task. □

When we want to show the completeness of the inference rules A1 to A7, we have to show the completeness of the inference rules C1 to C5 under class inclusion constraints, onto constraints and path functional dependencies first. Proving this completeness should follow the lines of the completeness proof of the inference rules C1 to C5 under only class inclusion constraints and onto constraints. The prerequisite is that the extension $\bigcup_{i\in\mathbb{N}} \operatorname{extrev}^i_\Upsilon(f)$ is an instance of a schema $D$ with path functional dependencies. Unfortunately, this is not true in general, although every S-tree satisfies all path functional dependencies trivially, because only one object is introduced for every path function. The problem arises when we want to remedy violations of onto constraints, because then we can no longer guarantee that a way description occurs only once as way-label. For example, when we consider a schema with the following components

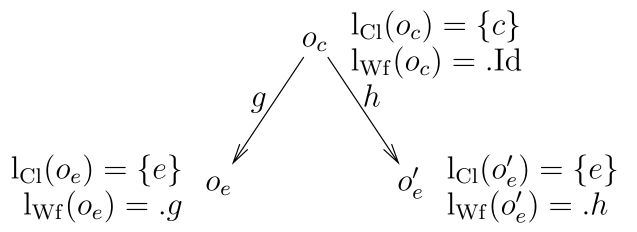
$$\begin{aligned}
\text{HIER}_D &= \{c'::c\} \ , \\
\text{SIG}_D &= \{c[g \Rightarrow e; h \Rightarrow e], c'[h \Rightarrow f]\} \ , \text{ and} \\
\text{SC}_D &= \text{AX} \\
&\quad\cup \\
&\quad \big\{c(.g \to .h), c'\{g|e\}\big\} \ , \text{ and}
\end{aligned}$$

we build the S-tree for the schema $(D \backslash \text{SC}_D) \cup \text{AX}$ and the sets $\{c\}$ and $\big\{c'\{g|e\}\big\}$, the resulting S-tree $t$ (Fig. 4.12(a)) satisfies the path functional dependency $c(.g \to .h)$. Alas, applying the operation $\operatorname{extrev}_{\{c'\{g|e\}\}}$ to this S-tree results in an extension (Fig. 4.12(b)) that does not satisfy the path functional dependency $c(.g \to .h)$ because of the objects $o_c$ and $o'_c$.

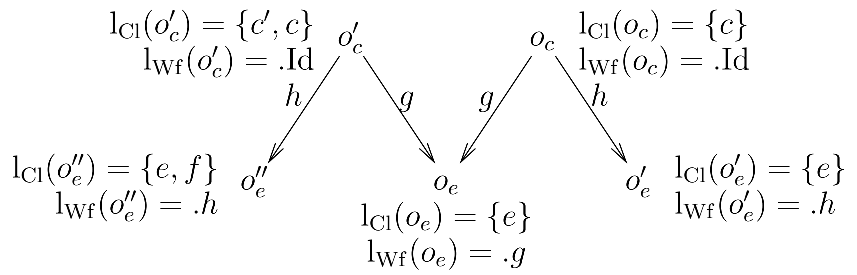
If we want to cure this violation, we might think of tampering with the class memberships of the object $o_c$ instead of introducing a new object, i. e., we might be tempted to make the object $o_c$ a member of the class $c'$. But this is not a good idea, because then the resulting extension satisfies semantic constraints the original S-tree does not satisfy. A different solution might be to stipulate the object $o'_e$ as value for the attribute $h$ of the object $o'_c$. But again we have to tamper with the class memberships this time of the object $o'_e$ because of the signature $c'[h \Rightarrow f]$ and incur the same problems as before.

We take more drastic precautions by simply not allowing such situations in that we demand that whenever an onto constraint $d\{m|c\}$ over a schema is declared, this attribute $m$ is a proper attribute for the class $d$.

Definition 4.49 (Proper Constraints)
*Let $\Xi$ be a set of constraints over a schema $D$. The set $\Xi$ of constraints is a* proper set of constraints *if for each onto constraint $d\{m|c\} \in \Xi$ the attribute $m$ is a proper attribute for the class $d$.*



(a) An S-tree satisfying path functional dependencies



(b) A changed S-tree violating path functional dependencies

Figure 4.12: Reasons for the restriction on onto constraints

The next lemma shows that when we restrict the set of constraints to proper sets of constraints, situations as mentioned above, i. e. the violation of path functional dependencies, do not occur. This is because whenever an object $o$ is referenced by an object via an attribute $m$, the reparation of violations of onto constraints never introduces another object referencing the object $o$ via the attribute $m$.

LEMMA 4.50
*Let $f$ be an instance of a schema $D$. Let $\Xi$ be a set of proper constraints over the schema $D$. Whenever* $\mathrm{ob}_f \models o'[m \rightarrow o]$, $c \in l_{Cl}(o)$ *and* $c'\{m|c\} \in \Xi^{+_D}$ *holds, then* $c' \in l_{Cl}(o')$.

PROOF. The onto constraint $c'\{m|c\} \in \Xi^{+_D}$ can only be derived by the inference rules C4 and C5 when an onto constraint $d'\{m|d\} \in \Xi$ exists such that $d' \subset c'$ and $c \subset d$ can be derived from the set $\Xi$ of constraints. But then the attribute $m$ is a proper attribute for the class $d'$, $\mathrm{SIG}_D \models d'[m \Rightarrow ()]$. Because of the well-typedness axioms there exists a signature atom $\mathrm{HIER}_D \cup \mathrm{SIG}_D \models e[m \Rightarrow ()]$ that covers $o'[m \rightarrow o]$, hence $e \in l_{Cl}(o')$, and therefore, because of the fact that the attribute $m$ is a proper attribute for the class $d'$, the class $e$ is a subclass of the class $d'$, $\mathrm{HIER}_D \models e{::}d'$. Then we derive $e \subset d'$, and hence $e \subset c'$ by transitivity, which implies due to the correctness of the inference rules $c' \in l_{Cl}(o')$.

□

In the sequel we deal only with proper sets of constraints without saying this explicitly.

The next corollary draws the connection between the preceding lemma and the operation extrev, which can be reused because the operation unsat can be applied to a set of constraints as well.

COROLLARY 4.51

*Let $f$ be an instance of a schema $D$. Let $\Xi$ be a set of proper constraints over the schema $D$. If $(C, o, m) \in \mathrm{unsat}(f, \Xi)$ and $\mathrm{ob}_f \models o'[m \to o]$, then the set $C$ is empty, $C = \emptyset$.*

To show the completeness of the inference rules C1 to C5 under class inclusion constraints, onto constraints and path functional dependencies, we exploit a property of extensions with labelling, which is in a sense a generalisation of property 2 of extensions with labelling for attribute value paths.

LEMMA 4.52

*Let $f$ be an instance of a schema $D$ with only the set $\mathrm{AX}$ as semantic constraints, $\mathrm{SC}_D = \mathrm{AX}$, with labelling based on a set $C \subset \mathrm{CLASS}_D$ of classes and a set $\Upsilon$ of class inclusion constraints and onto constraints over the schema $D$. Let $o \in \mathrm{obj}(f)$ be an object, $c \in \mathrm{l_{Cl}}(o)$ be a class, and $p \in \mathrm{PathFuncs}_{D\cup\Upsilon}(c)$ be a path function. Then there exists an object $o'' \in \mathrm{obj}(f)$ with $o.p = \{o''\}$ and $\mathrm{l_{Wf}}(o) \circ p = \mathrm{l_{Wf}}(o'')$.*

PROOF. We show this by induction on the length of the path function $p$.

- $\mathrm{len}(p) = 0$. Then $o.p = \{o\}$ according to Def. 4.26, and $\mathrm{l_{Wf}}(o) \circ .\mathrm{Id} = \mathrm{l_{Wf}}(o)$.
- $\mathrm{len}(p) > 0$. Then the path function $p$ is of the form $p = p' \circ .m$, and $o.p' = \{o'\}$ according to the inductive assumption. Since $f$ is an instance with labelling based on the sets $C$ and $\Upsilon$,

$$c \in \mathrm{l_{Cl}}(o) = \bigcup_{c' \in C \cap \mathrm{Dom}_{D\cup\Upsilon}(\mathrm{l_{Wf}}(o))} \mathrm{Ran}_{D\cup\Upsilon}(c', \mathrm{l_{Wf}}(o)) \ ,$$

and hence there exists a class $c' \in C \cap \mathrm{Dom}_{D\cup\Upsilon}(\mathrm{l_{Wf}}(o))$ with $c \in \mathrm{Ran}_{D\cup\Upsilon}(c', \mathrm{l_{Wf}}(o))$. Since $p \in \mathrm{PathFuncs}_{D\cup\Upsilon}(c)$ is a path function, there exist classes $g$ and $i$ such that $i \in \mathrm{Ran}_{D\cup\Upsilon}(c, p')$ and $\mathrm{HIER}_{D\cup\Upsilon} \cup \mathrm{SIG}_{D\cup\Upsilon} \models i[m \Rightarrow g]$. So $c' \in C \cap \mathrm{Dom}_{D\cup\Upsilon}(\mathrm{l_{Wf}}(o) \circ p')$ with $i \in \mathrm{Ran}_{D\cup\Upsilon}(c', \mathrm{l_{Wf}}(o) \circ p')$ by Lem. 4.25, and consequently due to the inductive assumption ($\mathrm{l_{Wf}}(o) \circ p' = \mathrm{l_{Wf}}(o')$) $c' \in C \cap \mathrm{Dom}_{D\cup\Upsilon}(\mathrm{l_{Wf}}(o'))$ with $i \in \mathrm{Ran}_{D\cup\Upsilon}(c', \mathrm{l_{Wf}}(o'))$, which means $i \in \mathrm{l_{Cl}}(o')$, because $f$ is an instance with labelling based on the sets $C$ and $\Upsilon$. Since $\mathrm{HIER}_{D\cup\Upsilon} \cup \mathrm{SIG}_{D\cup\Upsilon} \models i[m \Rightarrow g]$ holds, $\mathrm{HIER}_D \cup \mathrm{SIG}_D \models i[m \Rightarrow g]$ holds as well. This signature atom demands due to the unique name axioms and to the definedness axioms the existence of exactly one object $o''$ with

$$\mathrm{ob}_f \models o'[m \to o''] \ , \tag{4.10}$$

which leads to the conclusion $o.p = \{o''\}$. Additionally, (4.10) implies $\mathrm{l_{Wf}}(o') \circ .m = \mathrm{l_{Wf}}(o'')$ by property 2 of way-labellings. Finally, we have $\mathrm{l_{Wf}}(o) \circ p = \mathrm{l_{Wf}}(o) \circ (p' \circ .m) = (\mathrm{l_{Wf}}(o) \circ p') \circ .m = \mathrm{l_{Wf}}(o') \circ .m = \mathrm{l_{Wf}}(o'')$.

□

An immediate consequence of this lemma is presented in the next corollary.

Corollary 4.53
*Let $f$ be an instance of a schema $D$ with only the set* AX *of axioms as semantic constraints,* $\mathrm{SC}_D = \mathrm{AX}$, *with labelling based on a set $C \subset \mathrm{CLASS}_D$ of classes and a set $\Upsilon$ of class inclusion constraints and onto constraints over the schema $D$. For all objects $o, o' \in \mathrm{obj}(f)$, and for all classes $c \in \mathrm{l}_{\mathrm{Cl}}(o) \cap \mathrm{l}_{\mathrm{Cl}}(o')$ if $o.p = o'.p$ for $p \in \mathrm{PathFuncs}_{D\cup\Upsilon}(c)$, then $\mathrm{l}_{\mathrm{Wf}}(o) = \mathrm{l}_{\mathrm{Wf}}(o')$.*

Finally, we are ready to show that the extension $\bigcup_{i\in\mathbb{N}} \mathrm{extrev}^i_\Upsilon(t)$ for an S-tree $t$ satisfies even the path functional dependencies in a schema and therefore is the counterexample needed to prove the completeness of the inference rules C1 to C5 under class inclusion constraints, onto constraints and path functional dependencies.

Lemma 4.54
*Let $D$ be a schema, $\Xi := \mathrm{CIC}_D \cup \mathrm{OC}_D \cup \mathrm{PFD}_D$ be the set of constraints in the schema $D$, and $t$ be an S-tree of the schema $D\backslash\Xi$ based on a set $C \subset \mathrm{CLASS}_D$ of classes and the set $\Upsilon := \mathrm{CIC}_D \cup \mathrm{OC}_D$. Then $\bigcup_{i\in\mathbb{N}} \mathrm{extrev}^i_\Upsilon(t)$ is an instance of the schema $D$.*

Proof. From Lem. 4.40, we know that $f := \bigcup_{i\in\mathbb{N}} \mathrm{extrev}^i_\Upsilon(t)$ is an instance of the schema $D\backslash\mathrm{PFD}_D$. So we deal now with the satisfaction of the path functional dependencies in the set $\mathrm{PFD}_D$.

Let $c \in \mathrm{CLASS}_D$ be a class and $o, o' \in \mathrm{obj}(f)$ be two objects such that $c \in \mathrm{l}_{\mathrm{Cl}}(o)$ and $c \in \mathrm{l}_{\mathrm{Cl}}(o')$, and $o.p = o'.p$ for some path function $p \in \mathrm{PathFuncs}_{D\backslash\mathrm{PFD}_D}(c)$. According to Cor. 4.53 it follows that $\mathrm{l}_{\mathrm{Wf}}(o) = \mathrm{l}_{\mathrm{Wf}}(o')$. The generation of the S-tree $t$ produces only one object for each way-label, and the operation $\mathrm{extrev}_\Upsilon$ retains this invariant due to Cor. 4.51, hence $o = o'$. Therefore every path functional dependency in the set $\mathrm{PFD}_D$ is trivially satisfied.

□

Now we can prove the completeness of the inference rules C1 to C5 under class inclusion constraints, onto constraints and path functional dependencies. The proof will be conducted by contraposition. We assume that we cannot derive a class inclusion constraint $\upsilon$ or onto constraint $\upsilon$ from the set $\Xi$ of constraints in a schema $D$, $\Xi \not\vdash_D \upsilon$. Then we construct an instance of the schema $D$ that is not an instance of the schema $D \cup \{\upsilon\}$.

Theorem 4.55 (Completeness of C1 to C5 under CICs, OCs and PFDs)
*Let $D$ be a schema, and $\upsilon$ be a class inclusion constraint or onto constraint over the schema $D$ such that $\upsilon$ cannot be derived from the set $\Xi := \mathrm{CIC}_D \cup \mathrm{OC}_D \cup \mathrm{PFD}_D$ of constraints in the schema $D$, $\Xi \not\vdash_D \upsilon$. Then there exists an instance $f$ of the schema $D$ such that $f$ is not an instance of the schema $D \cup \{\upsilon\}$.*

Proof. The proof of Theor. 4.41 can be used nearly unchanged. We only revert to Lem. 4.54 instead of Lem. 4.40. □

Knowing that the inference rules C1 to C5 are even complete under class inclusion constraints, onto constraints and path functional dependencies, we give an outline of how we prove the completeness of the inference rules A1 to A7. The basic idea is to show the completeness by contraposition. So instead of showing for a path functional dependency $c(X \rightarrow Y)$ that if every instance of a schema $D$ is an instance of the schema $D \cup \{c(X \rightarrow Y)\}$, then the path functional dependency is derivable from the set $\mathrm{CIC}_D \cup \mathrm{OC}_D \cup \mathrm{PFD}_D$, we construct an instance of the schema $D$ not satisfying the path functional dependency $c(X \rightarrow Y)$, if the dependency $c(X \rightarrow Y)$ is not derivable from the set $\mathrm{CIC}_D \cup \mathrm{OC}_D \cup \mathrm{PFD}_D$.

In principle, the construction of the counterexample exploits two ideas. The core of the counterexample generalises the well-known Armstrong construction for similar proofs in the relational data model, using two tuples agreeing exactly on the closure of a set of attributes $X$. In our case we use two objects of the class $c$. But, in general, two objects are not sufficient, so we take two S-trees with their roots being members of the class $c$. Then these roots have to agree exactly on their values for path functions in the closure $X^{+_{D,c}}$, because we want the roots to violate the path functional dependency $c(X \rightarrow Y)$. Thus we merge the two S-trees by removing some nodes, exactly those whose way-labels appear in the closure $X^{+_{D,c}}$, from one of the S-trees and afterwards by inserting missing attribute values for objects in the S-tree whose nodes got removed. These attribute values are objects of the S-tree that is still intact and carry the corresponding way-labels as the removed objects. The result, called *Two-S-graph*, satisfies the path functional dependencies in the schema $D$, but not the path functional dependency $c(X \rightarrow Y)$ (Lem. 4.64) and thus forms the prototype of the counterexample.

A Two-S-graph looks like a "Siamese twin": the roots are the heads and the objects with way-labels in the closure $X^{+_{D,c}}$ are the limbs that are grown together.

The satisfaction of class inclusion constraints is reached by carefully choosing the class memberships of the objects in the Two-S-graph.

Again we face the problem that a Two-S-graph does not satisfy onto constraints and simply applying the operation extrev on a Two-S-graph does not suffice, because, in general, the outcome does not satisfy the derivable path functional dependencies. For that we have to merge objects being inserted in different pruned-S-trees. Looking only at the newly inserted objects suffices, because once two objects satisfy the path functional dependencies, the objects reachable from these objects by path functions remain unchanged (Lem. 4.67). We call the operation that first performs the operation extrev and then merges objects, merge (Def. 4.65).

As with the operation extrev, applying the operation merge leads to new violations of onto constraints, thus we iterate the process over and over again. The final outcome is then the counterexample we are looking for: It is an instance of the schema $D$ but does not satisfy the path functional dependency $c(X \rightarrow Y)$.

The outcome of the iteration has still some similarity with the initial Two-S-graph, which looks like a "Siamese twin". First of all it is possible to reach every object by following the object's way-label starting from one of the roots. We call this property *root-reachability* (Def. 4.56). Secondly, there are only two roots, and, thirdly, if we reach one object by applying the object's way-label to one of the roots, there exists another object reachable from the other root by applying the first object's way-label to this other root. This property is called *root-isomorphism* (Def. 4.59). These two properties are exploited later on.

We begin with the definition of root-reachability, which is a property of instances with labelling being invariant under the transformation merge.

Definition 4.56 (Root-Reachability)
*Let $D$ be a schema, and $f$ be an instance of the schema $D\backslash(\mathrm{CIC}_D \cup \mathrm{OC}_D \cup \mathrm{PFD}_D)$ with labelling based on a set $C \subset \mathrm{CLASS}_D$ of classes and the set $\mathrm{CIC}_D \cup \mathrm{OC}_D$. The instance $f$ is* root-reachable*, if for all objects $o \in \mathrm{obj}(f)$ there exists a root $r \in \mathrm{Root}(f)$ such that $r.\mathrm{l_{Wf}}(o) = \{o\}$.*

A root-reachable instance has the property that whenever we apply an appropriate way description to a root, we get at most one object as result. But before we show this property, we prove whenever an object in an instance with labelling is reachable from a root by an appropriate way description, this very way description is the object's way-label.

Lemma 4.57
*Let $f$ be an instance of a schema $D$ with only the set $\mathrm{AX}$ of axioms as semantic constraints, $\mathrm{SC}_D = \mathrm{AX}$, with labelling based on a set $C \subset \mathrm{CLASS}_D$ of classes and a set $\Upsilon$ of class inclusion constraints and onto constraints over the schema $D$. We have that for all objects $o \in \mathrm{obj}(f)$, for all roots $r \in \mathrm{Root}(f)$, for all classes $c \in \mathrm{l_{Cl}}(r)$, and for all way descriptions $w \in \mathrm{WayDes}_{D\cup\Upsilon}(c)$, if $o \in r.w$, then $\mathrm{l_{Wf}}(o) = w$.*

Proof. We show this by induction on the length of way description $w$.

$\mathrm{len}(w) = 0$. This means that $w = .\mathrm{Id}$ and therefore $\mathrm{l_{Wf}}(r) = .\mathrm{Id}$.

$\mathrm{len}(w) > 0$. This means $w$ is of the form $w' \circ \pi m$ where $\pi \in \{., -\}$ and $m \in \mathrm{METH}_D$. Let $o \in r.w$. We distinguish two cases according to the structure of $\pi$:

$\pi = .$, then there exists $o' \in r.w'$ with $\mathrm{ob}_f \models o'[m \to o]$ and $\mathrm{l_{Wf}}(o') = w'$ by the inductive hypothesis. Additionally, we know that $w'$ does not end in $-m$, because $w$ is a well-formed way description and is of the form $w = w'.m$. Then $\mathrm{l_{Wf}}(o) = \mathrm{l_{Wf}}(o') \circ .m = w' \circ .m = w'.m = w$ because of Def. 4.28.

$\pi = -$, then there exists $o' \in r.w'$ with $\mathrm{ob}_f \models o[m \to o']$ and $\mathrm{l_{Wf}}(o') = w'$ by the inductive hypothesis. Additionally, we know that $w'$ does not end in $.m$, because $w$ is a well-formed way description and is of the form $w = w'-m$. Now we assume two cases:

$\mathrm{l_{Wf}}(o) = y-m$, then $y = y \circ (-m \circ .m) = (y \circ -m) \circ .m = y-m \circ .m = \mathrm{l_{Wf}}(o) \circ .m \overset{\text{Def. 4.28}}{=} \mathrm{l_{Wf}}(o') = w'$ and therefore $\mathrm{l_{Wf}}(o) = w'-m = w$.

$\mathrm{l_{Wf}}(o) = y\pi'm'$, $\pi' \in \{., -\}$, $m' \in \mathrm{METH}_D$, $\pi'm' \neq -m$, then we know the following $w' = \mathrm{l_{Wf}}(o') \overset{\text{Def. 4.28}}{=} \mathrm{l_{Wf}}(o) \circ .m = y\pi'm' \circ .m = y\pi'm'.m$, which is a contradiction, because $w'$ does not end in $.m$.

□

LEMMA 4.58
*Let $D$ be a schema, and $f$ be a root-reachable instance of the schema $D\backslash(\mathrm{CIC}_D \cup \mathrm{OC}_D \cup \mathrm{PFD}_D)$ with labelling based on a set $C \subset \mathrm{CLASS}_D$ of classes and the set $\mathrm{CIC}_D \cup \mathrm{OC}_D$. Then for all roots $r \in \mathrm{Root}(f)$, for all classes $c \in \mathrm{l_{Cl}}(r)$, and for all way descriptions $w \in \mathrm{WayDes}_D(c)$: $|r.w| \leq 1$.*

PROOF. We show this by induction on the length of way description $w$.

**$\mathrm{len}(w) = 0$.** Trivial.

**$\mathrm{len}(w) = 1$.** This means that $w$ is of the form $\pi m$ where $\pi \in \{., -\}$ and $m \in \mathrm{METH}_D$. We consider two cases.

**$\pi = .$,** then $|r..m| = |\{o\}| = 1$ for the object $o$ with $\mathrm{ob}_f \models r[m \rightarrow o]$.

**$\pi = -$,** then we assume $o_1, o_2 \in r.-m$ with $o_1 \neq o_2$. We know $\mathrm{l_{Wf}}(o_1) = \mathrm{l_{Wf}}(o_2) = -m$ because of Lem. 4.57. Because of the root-reachability there exists $r' \in \mathrm{Root}(f)$ with $r'.-m = \{o_1\}$. Since $\{o_1, o_2\} \subset r.-m$ and $o_1 \neq o_2$, $r$ and $r'$ are different objects, $r \neq r'$. So we know $o_1 \in r.-m$ and $o_1 \in r'.-m$ as in Fig. 4.13. But then $\mathrm{ob}_f \models o_1[m \rightarrow r] \wedge o_1[m \rightarrow r']$, which is a contradiction to the scalarity of $m$. This means $|r.-m| \leq 1$.

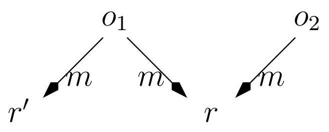


Figure 4.13: The situation that cannot arise in the case $\mathrm{len}(w) = 1$ and $\pi = -$

**$\mathrm{len}(w) = n + 1$, $n > 0$.** So $w$ is of the form $w'\pi m$ where $\pi \in \{., -\}$ and $m \in \mathrm{METH}_D$, and $|r.w'| \leq 1$ by the induction hypothesis. If $|r.w'| = 0$, then we know that $|r.w| = 0$. So the case that $|r.w'| = 1$ needs to be investigated. We consider two cases according to the structure of $\pi$.

**$\pi = .$,** then $|r.w'| = 1$, say $r.w' = \{o\}$, and $|r.w| = |o..m| = 1$.

**$\pi = -$,** then $|r.w'| = 1$, say $r.w' = \{o\}$, and $\mathrm{l_{Wf}}(o) \overset{\text{Lem.4.57}}{=} w'$. We assume now $o_1, o_2 \in r.w$ with $o_1 \neq o_2$. Again we know $\mathrm{l_{Wf}}(o_1) = \mathrm{l_{Wf}}(o_2) = w$ because of Lem. 4.57, and $\mathrm{ob}_f \models o_1[m \rightarrow o] \wedge o_2[m \rightarrow o]$. Then there exists a root $r' \in \mathrm{Root}(f)$ such that $r'.w = \{o_1\}$ because of the root-reachability of the instance $f$. Since $\{o_1, o_2\} \subset r.w$ and $o_1 \neq o_2$, $r$ and $r'$ are different objects, $r \neq r'$. Now $o \notin r'.w'$, because if we assume that $o \in r'.w'$ as in Fig. 4.14(a), then $\{o_1, o_2\} \in r'.w$, which is a contradiction. Then there exists $o_1'$ with $o_1' \in r'.w'$ as in Fig. 4.14(b), and therefore $\mathrm{ob}_f \models o_1[m \rightarrow o_1'] \wedge o_1[m \rightarrow o]$, which is a contradiction to the scalarity of $m$. This means $|r.w| \leq 1$.

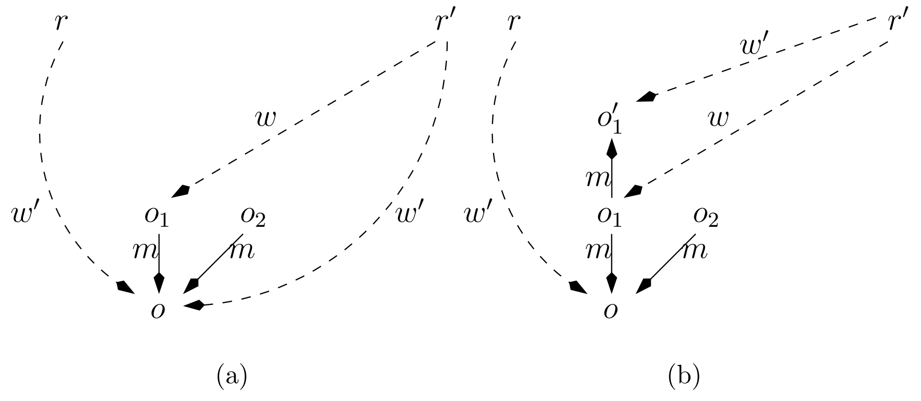


Figure 4.14: Situations that cannot arise in the case $\text{len}(w) = n + 1$ and $\pi = -$

□

As mentioned previously, the extensions obtained during the construction of the counterexample look the same which ever root we assume as vista point. This feature is captured by the property root-isomorphism. Again this property is invariant under the transformation merge.

Definition 4.59 (Root-Isomorphism)
*Let $D$ be a schema, and $f$ be a root-reachable instance of the schema $D\backslash(\text{CIC}_D \cup \text{OC}_D \cup \text{PFD}_D)$ with labelling based on a set $C \subset \text{CLASS}_D$ of classes and the set $\text{CIC}_D \cup \text{OC}_D$. The instance $f$ is* root-isomorphic *if*

- *the instance $f$ has at most two roots, $|\text{Root}(f)| \leq 2$, and*
- *for all roots $r \in \text{Root}(f)$, for all objects $o \in \text{obj}(f)$ with $r.\text{l}_{\text{Wf}}(o) = \{o\}$, and for all roots $r' \in \text{Root}(f)$ there exists an object $o' \in \text{obj}(f)$ such that $r'.\text{l}_{\text{Wf}}(o) = \{o'\}$.*

The prototype for the counterexample is a Two-S-graph, which we compose out of two S-trees, hence the name Two-S-graph. From one of these trees we remove objects and create links to the other S-tree by inserting objects of the other S-tree for missing attribute values.

Definition 4.60 (Two-S-Graph)
*Let $D$ be a schema, and $\Xi := \text{CIC}_D \cup \text{OC}_D \cup \text{PFD}_D$ be the set of constraints in the schema $D$. A* Two-S-graph *for the schema $D$ and a path functional dependency $c(X \to Y)$ over the schema $D$ is an extension $f = \langle \text{pop}_f | \text{ob}_f \rangle$ of the schema $D\backslash\Xi$ with labelling based on the sets $\{c\}$ and $\text{CIC}_D \cup \text{OC}_D$ constructed as follows.*

**Step 1:** *Construct two S-trees $f_1 := \langle \text{pop}_1 | \text{ob}_1 \rangle$ and $f_2 := \langle \text{pop}_2 | \text{ob}_2 \rangle$ of the schema $D\backslash\Xi$ with labelling based on the sets $\{c\}$ and $\text{CIC}_D \cup \text{OC}_D$ such that $\text{obj}(f_1) \cap \text{obj}(f_2) = \emptyset$. Let $R_1$, $R_2$ denote the single element in $\text{Root}(f_1)$, $\text{Root}(f_2)$, respectively.*

**Step 2:** *Remove any $v : c_v \in \text{pop}_2$ and $u[m \to v] \in \text{ob}_2$ whenever $\text{l}_{\text{Wf}}(v) \in X^{+_{D,c}}$.*

**Step 3:** $\mathrm{pop}_f := \mathrm{pop}_1 \cup \mathrm{pop}_2$ *and* $\mathrm{ob}_f := \mathrm{ob}_1 \cup \mathrm{ob}_2$.

**Step 4:** *For each* $u : c_u \in \mathrm{pop}_f$ *and* $m \in \mathrm{Attr}_D(c_u)$ *where due to Step 2* $\mathrm{ob}_f \not\models u[m \to w]$ *for all* $w \in \mathrm{obj}(f)$*, add* $u[m \to R_1.\mathrm{l_{Wf}}(u).m]$ *to* $\mathrm{ob}_f$.

Example 4.61
*The Two-S-graph (Fig. 4.15(b)) built for the schema in Exam. 4.46 and the path functional dependency* $h(.v \to .n)$ *is composed out of two S-trees (Fig. 4.15(a)). The closure of the set* $\{.v\}$ *contains the path function* $.v$*, therefore we remove the object* $x_d$ *and insert the object* $o_d$ *as attribute value for the object* $x_h$*. Because of the path functional dependency* $h(.v \to .n.u)$ *the path function* $.n.u$ *is an element of the closure* $\{.v\}^{+_{S,h}}$*, hence the removal of the object* $x'_h$ *and all objects reachable from the object via path functions.*

A Two-S-graph constructed for a schema satisfies at least the set AX of axioms.

Lemma 4.62
*Let* $g$ *be a Two-S-graph for a schema* $D$ *and a path functional dependency* $c(X \to Y)$ *over the schema* $D$*. The Two-S-graph* $g$ *is an instance of the schema* $D \backslash (\mathrm{CIC}_D \cup \mathrm{OC}_D \cup \mathrm{PFD}_D)$.

Proof. The two S-trees needed for the construction of the Two-S-graph $g$ are instances of the schema $D \backslash (\mathrm{CIC}_D \cup \mathrm{OC}_D \cup \mathrm{PFD}_D)$ according to Lem. 4.32. Because of the congruence replacing some objects of one S-tree by objects of the other S-tree does not lead to a violation of the unique name axioms, well-typedness axioms or definedness axioms. □

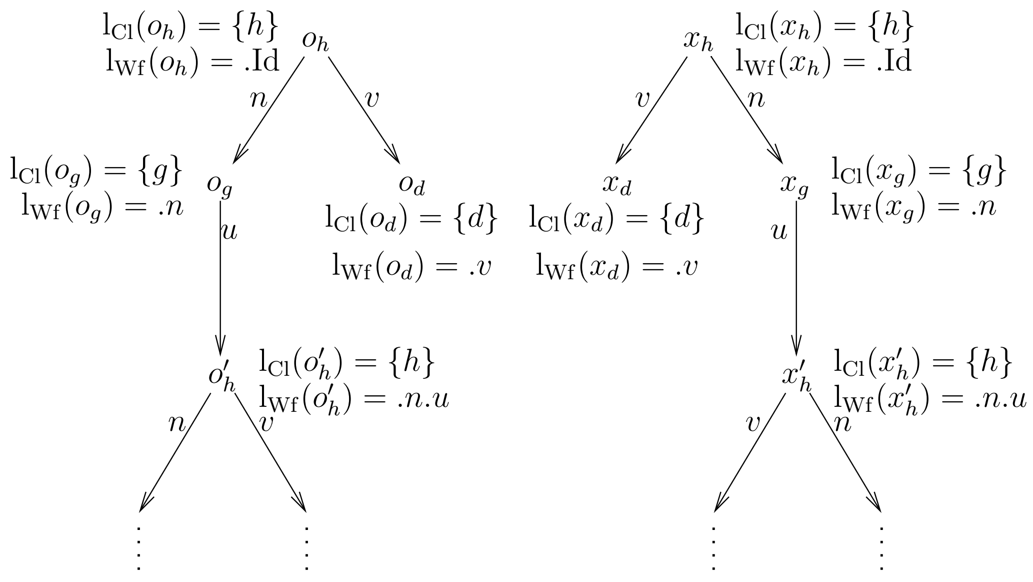
The construction of a Two-S-graph ensures that only those objects are removed whose way-labels are elements of the closure of the corresponding set of path functions.

Corollary 4.63
*Let* $g$ *be a Two-S-graph for a schema* $D$ *and a path functional dependency* $c(X \to Y)$ *over the schema* $D$*. Then for all objects* $o \in \mathrm{obj}(g)$*,* $R_1.\mathrm{l_{Wf}}(o) = R_2.\mathrm{l_{Wf}}(o) = o$ *iff* $\mathrm{l_{Wf}}(o) \in X^{+_{D,c}}$.

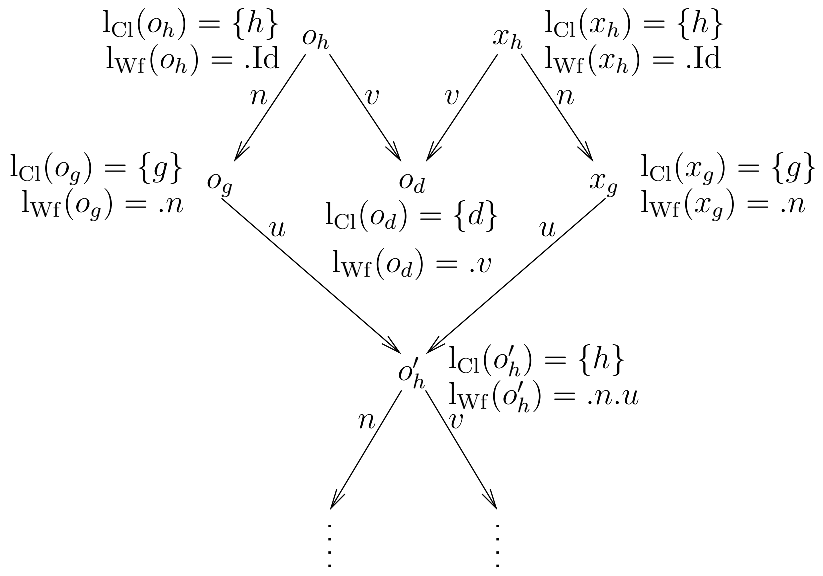
A Two-S-graph is indeed a prototype of the counterexample in that a Two-S-graph potentially satisfies the derivable path functional dependencies but not the path functional dependency the Two-S-graph is constructed for. Additionally, a Two-S-graph looks like a "Siamese twin", i. e., it is root-isomorphic.

Lemma 4.64
*Let* $D$ *be a schema,* $\Xi := \mathrm{CIC}_D \cup \mathrm{OC}_D \cup \mathrm{PFD}_D$ *be the set of constraints in the schema* $D$*,* $c(X \to Y) \notin \Xi^{+_D}$ *be a path functional dependency not derivable from the set* $\Xi$*. Let* $g$ *be a Two-S-graph for the schema* $D$ *and the path functional dependency* $c(X \to Y)$.

1. *The Two-S-graph* $g$ *is root-isomorphic.*
2. *(a) The Two-S-graph* $g$ *is an instance of the schema* $(D \backslash \Xi) \cup \Xi^{\oplus_D}$.



(a) The two S-trees for the Two-S-graph



(b) The Two-S-graph

Figure 4.15: How to construct the Two-S-graph in Exam. 4.61

(b) *The Two-S-graph $g$ is not an instance of the schema $(D\backslash\Xi) \cup \Xi^{\oplus_D} \cup \{c(X \rightarrow Y)\}$.*

PROOF. 1. The Two-S-graph is an instance of the schema $D\backslash\Xi$ according to

Lem. 4.62.

For each object $o$ inserted in one of the S-trees $f_1$ or $f_2$ during the construction of the Two-S-graph $g$, we know that the way-label $\mathrm{l_{Wf}}(o)$ is a path function. Without loss of generality, we assume that $o$ has been inserted into the S-tree $f_1$. Then there exists an object $o'$ with $R_1.\mathrm{l_{Wf}}(o) = \{o'\}$ and $\mathrm{l_{Wf}}(o') \overset{\text{Lem.4.57}}{=} \mathrm{l_{Wf}}(R_1) \circ \mathrm{l_{Wf}}(o) = .\mathrm{Id} \circ \mathrm{l_{Wf}}(o) = \mathrm{l_{Wf}}(o)$; but only one object is introduced for each S-tree and path function, and therefore $o = o'$. The removal of nodes and edges from one of the S-trees and the addition of missing links does not mar this property, and thus the Two-S-graph is root-reachable.

A similar argumentation holds for the property root-isomorphism. The introduction of objects in the S-trees $f_1$ and $f_2$ during the construction of the Two-S-graph $g$ is executed according to the set of path functions starting at the class $c$. So whenever an object is reachable from a root of the Two-S-graph $g$ either this object is reachable by the other root as well or the object with the same way-label. Therefore the Two-S-graph $g$ is root-isomorphic, because the Two-S-graph $g$ contains only the roots $R_1$ and $R_2$ by definition.

2. (a) Let

$$c'(p_1 \cdots p_m \rightarrow p_{m+1} \cdots p_n) \in \Xi^{+_D} \tag{4.11}$$

be a path functional dependency, $o, o' \in \mathrm{obj}(g)$ be two objects such that $c' \in \mathrm{l_{Cl}}(o) \cap \mathrm{l_{Cl}}(o')$, and

$$o.p_i = o'.p_i \text{ for all } i \in \{1, \ldots, m\} \ . \tag{4.12}$$

Then $w := \mathrm{l_{Wf}}(o) = \mathrm{l_{Wf}}(o')$ because of Cor. 4.53. We consider two cases:

i. There exists a root $r \in \mathrm{Root}(g)$ such that $o, o' \in r.w$. Then $o = o'$ because $g$ is root-reachable, and therefore $|r.w| \leq 1$ (Lem. 4.58).

ii. There does not exist a root $r \in \mathrm{Root}(g)$ such that $o, o' \in r.w$. Then there are roots $r, r' \in \mathrm{Root}(g)$ with $r.w = o \neq o' = r'.w$. Because of $c' \in \mathrm{l_{Cl}}(o)$ and $g$ is an extension with labelling, $c' \in \mathrm{Ran}_D(c, w)$, and therefore we can derive

$$c(w \circ p_1 \cdots w \circ p_m \rightarrow w \circ p_{m+1} \cdots w \circ p_n) \tag{4.13}$$

from (4.11) by simple prefix augmentation. Because of (4.12) $r.w \circ p_i = r'.w \circ p_i$ for $i \in \{1, \ldots, m\}$. By Cor. 4.63, it follows that $w \circ p_i \in X^{+_{D,c}}$ for $i \in \{1, \ldots, m\}$. This means $w \circ p_j \in X^{+_{D,c}}$ for $j \in \{m+1, \ldots, n\}$ because of (4.13). Consequently, $r.w \circ p_j = r'.w \circ p_j$ for $j \in \{m+1, \ldots, n\}$, and hence $o.p_j = o'.p_j$ for $j \in \{m+1, \ldots, n\}$, which means (4.11) is satisfied.

(b) The Two-S-graph $g$ does not satisfy $c(X \rightarrow Y)$ because $c(X \rightarrow Y) \notin \Xi^{+_D}$, and therefore there is $p \in Y \backslash X^{+_{D,c}}$, hence $R_1.p \neq R_2.p$ by Cor. 4.63.

□

Having the prototype of the counterexample, we deal with the satisfaction of onto constraints. The basic idea is to insert a pruned-S-tree whenever an object violates an onto constraint. Unfortunately, this insertion entails in general a violation of path functional dependencies, which we cure by removing objects from except from one all of the inserted pruned-S-trees and adding missing links.

Definition 4.65 (Merge)
*Let $D$ be a schema, $\Xi := \mathrm{CIC}_D \cup \mathrm{OC}_D \cup \mathrm{PFD}_D$ be the set of constraints in the schema $D$, and $\Upsilon := \mathrm{CIC}_D \cup \mathrm{OC}_D$ be the set of class inclusion constraints and onto constraints in the schema $D$. Let $f$ be a root-isomorphic instance of the schema $D\backslash\Xi$ with labelling based on a set $C \subset \mathrm{CLASS}_D$ of classes and the set $\Upsilon$. The transformation* $\mathrm{merge}_D(f)$ *is defined as the outcome of the following steps.*

*Let $r \in \mathrm{Root}(f)$ be an arbitrary root. We define*

$$\mathrm{close} = \{u \in \mathrm{obj}(f) \mid \textit{ex. } r' \in \mathrm{Root}(f)\backslash\{r\}\colon\ r.\mathrm{l}_{\mathrm{Wf}}(u) = r'.\mathrm{l}_{\mathrm{Wf}}(u)\} \ .$$

*For every object $r''$ that is the root of a pruned-S-tree* $\mathrm{pst}(C', o, m, \mathrm{l}_{\mathrm{Wf}}(o))$ *newly inserted in* $\mathrm{extrev}_\Upsilon(f)$*, and that is reachable from $r' \in \mathrm{Root}(f)\backslash\{r\}$ ($r'' \in r'.\mathrm{l}_{\mathrm{Wf}}(r'')$) and not reachable from $r$ ($r'' \notin r.\mathrm{l}_{\mathrm{Wf}}(r'')$), we define*

$$\pi(r'') = \{p \in \bigcup_{d \in \mathrm{l}_{\mathrm{Cl}}(r'')} \mathrm{PathFuncs}_D(d) \mid r''.p \subset \mathrm{close}\} \quad ,$$

*and*

$$\Pi(r'') = \bigcup_{d \in \mathrm{l}_{\mathrm{Cl}}(r'')} \left( \bigcup_{\emptyset \neq N \subset \pi(r'') \cap \mathrm{PathFuncs}_D(d) \textit{ and } |N| \in \mathbb{N}} N^{+_{D,d}} \right) \ .$$

*Then we apply the following steps to $f' := \mathrm{extrev}_\Upsilon(f)$.*

1. *For each $p \in \Pi(r'')\backslash\pi(r'')$ remove the object $u$ with $r''.p = \{u\}$ and any incident edge into $u$ from $f'$.*
2. *For each remaining object $o'$, class $d \in \mathrm{l}_{\mathrm{Cl}}(o')$, and attribute $n \in \mathrm{Attr}_D(d)$ where due to the first step $\mathrm{ob}_f \not\models o'[n \to o'']$ for all remaining objects $o''$ add $o'[n \to r.\mathrm{l}_{\mathrm{Wf}}(o') \circ .m]$.*

We designate extensions satisfying the derivable path functional dependencies as input for the operation merge. As the set $X$ of path functions was taken in the construction of a Two-S-graph to determine which objects had to be removed, the set $\pi(r'')$ for the root $r''$ of a newly inserted pruned-S-tree is used for this job as well. Applying a path function from the set $\pi(r'')$ to the root $r''$ yields an object that is a member of the set close, which comprises all objects reachable from all roots.

Example 4.66
*The Two-S-graph in Exam. 4.61 violates the onto constraint $e\{k|h\}$ for two objects, $o_h$ and $x_h$. So we insert for both of them a pruned-S-tree, but this extension (Fig. 4.16(a)) violates the path functional dependency $e(.k.v \to .j)$ derivable from the set of constraints*

*in the schema $S$. We remedy this violation by removing the object $x_f$ and stipulating the object $o_f$ as value for the attribute $j$ of the object $x_e$ (Fig. 4.16(b)).*

*The extension in Fig. 4.16(b) does not satisfy the onto constraint $f''\{p|e\}$, which leads to another application of the operation* merge *on the outcome. In this application two objects $o'_f$ and $x'_f$ are introduced, which violate the path functional dependency $f''(.p.j \to .q)$. But this violation is repaired due to the fact that $o_f \in \mathrm{close}$ and hence $.p.j \in \pi(x'_f)$, and so, finally, $.q \in \pi(x'_f)^{+_{S,f''}}$.*

The removal of objects in the operation merge leaves the objects in the original instance untouched.

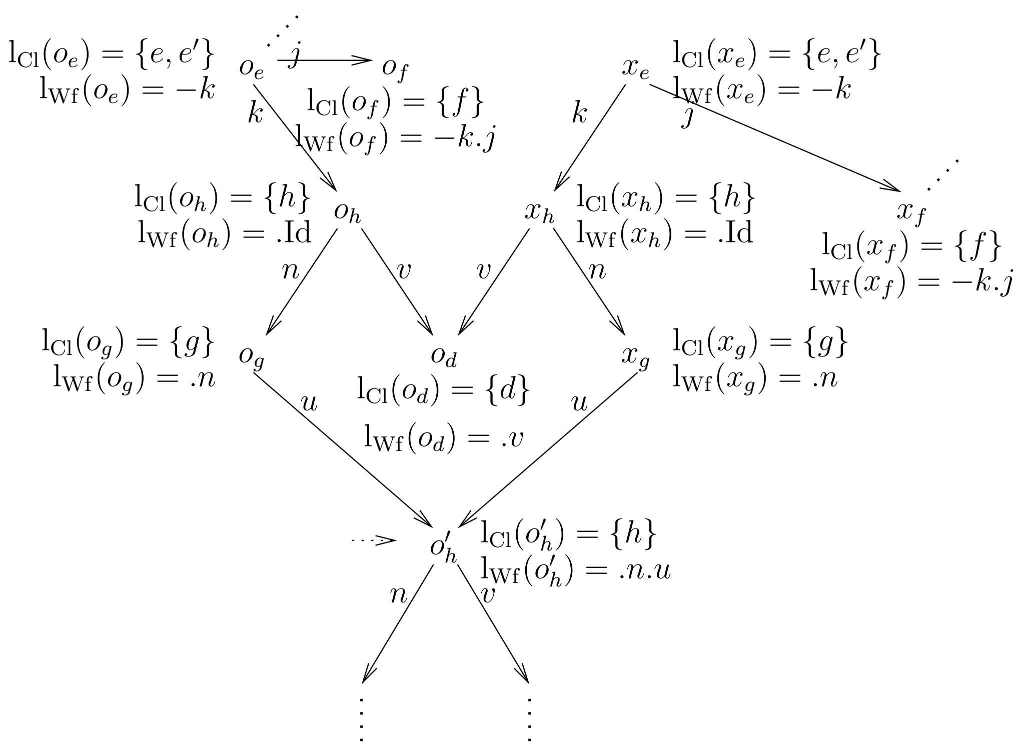
Lemma 4.67
*Let $D$ be a schema, $\Xi := \mathrm{CIC}_D \cup \mathrm{OC}_D \cup \mathrm{PFD}_D$ be the set of constraints in the schema $D$, and $\Upsilon := \mathrm{CIC}_D \cup \mathrm{OC}_D$ be the set of class inclusion constraints and onto constraints in the schema $D$. Let $f$ be a root-isomorphic instance of the schema $D\backslash\Xi$ with labelling based on a set $C \subset \mathrm{CLASS}_D$ of classes and the set $\Upsilon$, and an instance of the schema $(D\backslash\Xi)\cup\Xi^{\oplus_D}$. In the construction process of $\mathrm{merge}_D(f)$ no object $o \in \mathrm{obj}(f)$ is removed, i. e. for all objects $o \in \mathrm{obj}(f)$, $o \in \mathrm{obj}(\mathrm{merge}_D(f))$.*

Proof. We show $\bigcup_{p\in\Pi(r'')\backslash\pi(r'')} r''.p \cap \mathrm{obj}(f) = \emptyset$ for the root $r''$ of any pruned-S-tree $\mathrm{pst}(C', o, m, \mathrm{l}_{\mathrm{Wf}}(o))$ newly inserted in $\mathrm{extrev}_\Upsilon(f)$. Let $r' \neq r$ be a root of the extension $f'$ in Def. 4.65 with $r'.\mathrm{l}_{\mathrm{Wf}}(r'') = \{r''\}$.

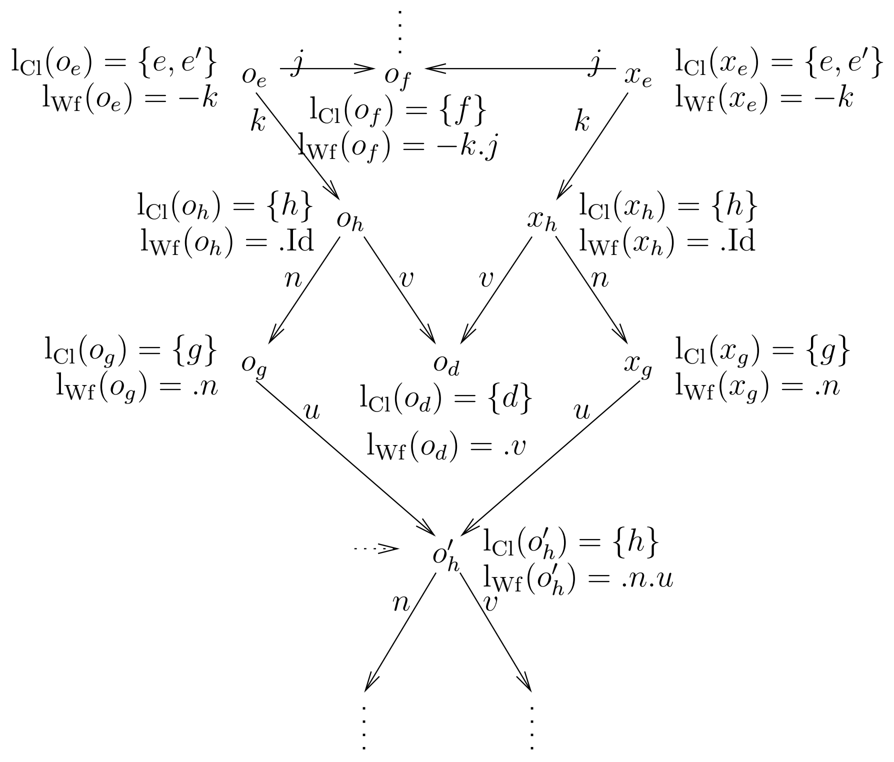
Let us conversely assume $v \in r''.p \cap \mathrm{obj}(f)$ for a $p \in \Pi(r'')\backslash\pi(r'')$. Then $v \notin \mathrm{close}$ by definition. Because of $r'.\mathrm{l}_{\mathrm{Wf}}(r'') \circ p = \{v\}$ and the root-isomorphism there exists $v' \in r.\mathrm{l}_{\mathrm{Wf}}(r'') \circ p \cap \mathrm{obj}(f)$ distinct from $v$, $v \neq v'$. Every path function in $\pi(r'')$ has got the prefix $.m$ due to its definition. The path function $p$ has got the prefix $.m$ since $v \in \mathrm{obj}(f)$. $p \in \Pi(r'')$ and therefore $d(N \to p) \in \Xi^{+_D}$ for a class $d \in \mathrm{l}_{\mathrm{Cl}}(r'')$ with $m \in \mathrm{Attr}_D(d)$ and some finite, non-empty set of path functions $N \subset \pi(r'')\cap\mathrm{PathFuncs}_D(d)$. The root $r''$ was added because of $d''\{m|d'\} \in \Upsilon^{+_D}$ for some class $d''$ with $d'' \subset d$ and for some class $d' \in \mathrm{l}_{\mathrm{Cl}}(o)$. Then $d''(N \to p) \in \Xi^{+_D}$ holds by path functional dependency inheritance. Due to simple prefix reduction, this entails $d'(\alpha(.m, N) \to \alpha(.m, \{p\})) \in \Xi^{+_D}$ with $\alpha(P, S) := \{t \mid Pt \in S\}$. But then $o$ and $r.\mathrm{l}_{\mathrm{Wf}}(o)$ violate this path functional dependency, because $o \neq r.\mathrm{l}_{\mathrm{Wf}}(o)$, $o.q = r.\mathrm{l}_{\mathrm{Wf}}(o) \circ q$ for each $q \in \alpha(.m, N)$ and $o.p' = v \neq v' = r.\mathrm{l}_{\mathrm{Wf}}(o) \circ p'$ for $\{p'\} = \alpha(.m, \{p\})$, which is a contradiction to the fact that $f$ is an instance of $(D\backslash\Xi) \cup \Xi^{\oplus_D}$. □

A Two-S-graph forms the prototype for the counterexample, i. e., the Two-S-graph is a “Siamese twin”, and satisfies the derivable path functional dependencies but not the path functional dependency it is constructed for. The operation merge treats these properties as invariants.

Lemma 4.68
*Let $D$ be a schema, $\Xi := \mathrm{CIC}_D \cup \mathrm{OC}_D \cup \mathrm{PFD}_D$ be the set of constraints in the schema $D$, $\Upsilon := \mathrm{CIC}_D \cup \mathrm{OC}_D$ be the set of class inclusion constraints and onto constraints in the schema $D$, and $c(X \to Y) \notin \Xi^{+_D}$ be a path functional dependency. Let $f$ be a*


$\mathrm{l_{Cl}}(o_e) = \{e, e'\}$
$\mathrm{l_{Wf}}(o_e) = -k$
$o_e$
$j$
$o_f$
$\mathrm{l_{Cl}}(o_f) = \{f\}$
$\mathrm{l_{Wf}}(o_f) = -k.j$
$k$
$x_e$
$\mathrm{l_{Cl}}(x_e) = \{e, e'\}$
$\mathrm{l_{Wf}}(x_e) = -k$
$k$
$j$
$\mathrm{l_{Cl}}(o_h) = \{h\}$
$\mathrm{l_{Wf}}(o_h) = .\mathrm{Id}$
$o_h$
$x_h$
$\mathrm{l_{Cl}}(x_h) = \{h\}$
$\mathrm{l_{Wf}}(x_h) = .\mathrm{Id}$
$x_f$
$\mathrm{l_{Cl}}(x_f) = \{f\}$
$\mathrm{l_{Wf}}(x_f) = -k.j$
$n$
$v$
$v$
$n$
$\mathrm{l_{Cl}}(o_g) = \{g\}$
$\mathrm{l_{Wf}}(o_g) = .n$
$o_g$
$o_d$
$\mathrm{l_{Cl}}(o_d) = \{d\}$
$\mathrm{l_{Wf}}(o_d) = .v$
$x_g$
$\mathrm{l_{Cl}}(x_g) = \{g\}$
$\mathrm{l_{Wf}}(x_g) = .n$
$u$
$u$
$o'_h$
$\mathrm{l_{Cl}}(o'_h) = \{h\}$
$\mathrm{l_{Wf}}(o'_h) = .n.u$
$n$
$v$

(a) First applying the operation extrev, . . .


$\mathrm{l_{Cl}}(o_e) = \{e, e'\}$
$\mathrm{l_{Wf}}(o_e) = -k$
$o_e$
$j$
$o_f$
$j$
$x_e$
$\mathrm{l_{Cl}}(x_e) = \{e, e'\}$
$\mathrm{l_{Wf}}(x_e) = -k$
$k$
$\mathrm{l_{Cl}}(o_f) = \{f\}$
$\mathrm{l_{Wf}}(o_f) = -k.j$
$k$
$\mathrm{l_{Cl}}(o_h) = \{h\}$
$\mathrm{l_{Wf}}(o_h) = .\mathrm{Id}$
$o_h$
$x_h$
$\mathrm{l_{Cl}}(x_h) = \{h\}$
$\mathrm{l_{Wf}}(x_h) = .\mathrm{Id}$
$n$
$v$
$v$
$n$
$\mathrm{l_{Cl}}(o_g) = \{g\}$
$\mathrm{l_{Wf}}(o_g) = .n$
$o_g$
$o_d$
$\mathrm{l_{Cl}}(o_d) = \{d\}$
$\mathrm{l_{Wf}}(o_d) = .v$
$x_g$
$\mathrm{l_{Cl}}(x_g) = \{g\}$
$\mathrm{l_{Wf}}(x_g) = .n$
$u$
$u$
$o'_h$
$\mathrm{l_{Cl}}(o'_h) = \{h\}$
$\mathrm{l_{Wf}}(o'_h) = .n.u$
$n$
$v$

(b) . . . , then merging objects

Figure 4.16: Applying the operation merge on the Two-S-graph in Fig. 4.15(b)

*root-isomorphic instance of the schema $D\backslash\Xi$ with labelling based on the sets $\{c\}$ and $\Upsilon$, such that the instance $f$ is an instance of the schema $(D\backslash\Xi) \cup \Xi^{\oplus_D}$ and not an instance of the schema $(D\backslash\Xi) \cup \Xi^{\oplus_D} \cup \{c(X \to Y)\}$.*

1. *The extension* $\mathrm{merge}_D(f)$ *is root-isomorphic.*
2. *The extension* $\mathrm{merge}_D(f)$ *is an instance of* $(D\backslash\Xi) \cup \Xi^{\oplus_D}$.
3. *The extension* $\mathrm{merge}_D(f)$ *is not an instance of* $(D\backslash\Xi) \cup \Xi^{\oplus_D} \cup \{c(X \to Y)\}$.

PROOF. 1. We show first that the extension $\mathrm{merge}_D(f)$ is root-reachable. Because of Lem. 4.67 and the root-reachability of the instance $f$, for each object $o \in \mathrm{obj}(f)$, there exists a root $r \in \mathrm{Root}(\mathrm{merge}_D(f))$ such that $r.\mathrm{l_{Wf}}(o) = \{o\}$.

So we consider now objects added in the construction of $\mathrm{merge}_D(f)$. Let $o \in \mathrm{obj}(f)$ be an object for which a pruned-S-tree $\mathrm{pst}(C', o, m, \mathrm{l_{Wf}}(o))$ has been introduced. Then the only access to an object in the pruned-S-tree is via the object $o$. But then there exists a root $r \in \mathrm{Root}(\mathrm{merge}_D(f))$ with $r.\mathrm{l_{Wf}}(o) = \{o\}$. The root $o'$ of the pruned-S-tree has $\mathrm{l_{Wf}}(o) \circ -m$ as way-label and $\mathrm{l_{Wf}}(o)$ does not end in $.m$ due to Cor. 4.51. This implies $r.\mathrm{l_{Wf}}(o') = \{o'\}$, and therefore $r.\mathrm{l_{Wf}}(o'') = \{o''\}$ for each object $o''$ in the pruned-S-tree, because $\mathrm{l_{Wf}}(o'') = \mathrm{l_{Wf}}(o') \circ p$ for some path function $p$. The removal of nodes and edges from pruned-S-trees and the addition of missing links does not impair this property, and thus the extension $\mathrm{merge}_D(f)$ is root-reachable.

The same lines of arguments can be followed when proving the property root-isomorphism.

2. The construction process for $\mathrm{merge}_D(f)$ does not lead to a violation of AX. So we have to look at the path functional dependencies in $\Xi^{\oplus_D}$.

By contradiction. Let

$$c'(p_1 \cdots p_m \to p) \tag{4.14}$$

be a path functional dependency derivable from $\Xi$ and $o, o' \in \mathrm{obj}(\mathrm{merge}_D(f))$ be two distinct objects violating this path functional dependency ($o.p_i = o'.p_i$ for $i \in \{1, \ldots, m\}$ implies $\mathrm{l_{Wf}}(o) = \mathrm{l_{Wf}}(o')$ by Cor. 4.53, and $o.p \neq o'.p$). Because of the root-isomorphism and $\mathrm{l_{Wf}}(o) = \mathrm{l_{Wf}}(o')$, the objects $o$ and $o'$ are either both old objects, $o, o' \in \mathrm{obj}(f)$, or newly inserted objects, $o, o' \in \mathrm{obj}(\mathrm{merge}_D(f))\backslash\mathrm{obj}(f)$. Since $f$ is an instance of $(D\backslash\Xi) \cup \Xi^{\oplus_D}$ and because of Lem. 4.67 two objects in $\mathrm{obj}(f)$ cannot violate (4.14), and so both objects must have been newly added in two distinct pruned-S-trees. Distinct because way-labels are unique in pruned-S-trees.

Let $u, u'$ be the corresponding roots of the pruned-S-trees involved. Let $e'\{a|d''\}$ be an onto constraint that gave rise to the introduction of the root $u$. The onto constraint $e'\{a|d''\}$ is derivable from a proper onto constraint $d'\{a|d''\}$ with $\Upsilon^{+_D} \vdash_D d' \subset e'$ and, by Lem. 4.50, gave rise to the introduction of the root $u$. Then the onto constraint $d'\{a|d''\}$ gave also rise to the introduction of the root $u'$ because of $o.p_i = o'.p_i$ for $i \in \{1, \ldots, m\}$ and the root-isomorphism of the extension $f$.

Since all onto constraints are proper and $\mathrm{l}_{\mathrm{Cl}}(u) = \mathrm{Ran}_{(D \setminus \Xi) \cup \Upsilon}(c, \mathrm{l}_{\mathrm{Wf}}(u))$, $\Upsilon^{+_D} \vdash_D d' \subset e$ holds for all classes $e \in \mathrm{l}_{\mathrm{Cl}}(u)$. Then there exists a unique $q \in \bigcup_{d \in \mathrm{l}_{\mathrm{Cl}}(u)} \mathrm{PathFuncs}_D(d)$ with $u.q = \{o\}$ and $u'.q = \{o'\}$. As a consequence $u$ and $u'$ violate

$$d'(q \circ p_1 \cdots q \circ p_m \to q \circ p) \tag{4.15}$$

for the class $d'$, which can be derived from (4.14) by iterated application of simple prefix augmentation.

Since $u \neq u'$ and the root-reachability of $\mathrm{merge}_D(f)$, there are distinct roots $r, r' \in \mathrm{Root}(\mathrm{merge}_D(f))$ such that $r.\mathrm{l}_{\mathrm{Wf}}(u) = \{u\}$ and $r'.\mathrm{l}_{\mathrm{Wf}}(u) = \{u'\}$. Without loss of generality let $r$ be the root chosen in the construction of $\mathrm{merge}_D(f)$. Then $\{q \circ p_1, \ldots, q \circ p_m\} \subset \Pi(u')$, because if $q \circ p_i \notin \Pi(u')$, then neither $u'.q \circ p_i \in$ close nor $u'.q \circ p_i$ is removed in the first step of the construction of $\mathrm{merge}_D(f)$ and therefore $o.p_i = u.q \circ p_i \neq u'.q \circ p_i = o'.p_i$, which contradicts $o.p_i = o'.p_i$.

Since there exist a finite, non-empty set of path-functions $N \subset \pi(u') \cap \mathrm{PathFuncs}_D(d')$ such that $q \circ p_i \in N^{+_{D,d'}}$ and the path functional dependency (4.15), $q \circ p \in N^{+_{D,d'}}$ holds.

If we assume that

- $q \circ p \in \pi(u')$, then $u.q \circ p = u'.q \circ p$ by definition of $\pi(u)$ and
- $q \circ p \in N^{+_{D,d'}} \setminus \pi(u')$, then $u'.q \circ p$ is removed in the first step of the construction of $\mathrm{merge}_D(f)$, and therefore by the second step of the construction $u.q \circ p = u'.q \circ p$.

The equality $u.q \circ p = u'.q \circ p$ contradicts that the path functional dependency (4.15) is violated. Therefore (4.14) cannot have been violated.

3. According to Lem. 4.67, the construction process does not touch the set $v \in \mathrm{obj}(f)$ of objects. Therefore the pair $o, o' \in \mathrm{obj}(f)$ of objects that violates $c(X \to Y)$ in $f$ still violates $c(X \to Y)$ in $\mathrm{merge}_D(f)$.

□

Finally, we show the completeness of the inference rules A1 to A7 by contraposition. So we assume conversely that we cannot derive a path functional dependency $c(X \to Y)$ from the set of constraints in a schema $D$. Then we construct an instance of the schema $D$ that does not satisfy the dependency $c(X \to Y)$.

Theorem 4.69 (Completeness of A1 to A7 under CICs, OCs and PFDs)
*Let $D$ be a schema, $\Xi := \mathrm{CIC}_D \cup \mathrm{OC}_D \cup \mathrm{PFD}_D$ be the set of constraints in the schema $D$, $\Upsilon := \mathrm{CIC}_D \cup \mathrm{OC}_D$ be the set of class inclusion constraints and onto constraints in the schema $D$, and $c(X \to Y) \notin \Xi^{+_D}$ be a path functional dependency over the schema $D$. Then there exists an instance $f$ of the schema $D$ that is not an instance of the schema $D \cup \{c(X \to Y)\}$.*

PROOF. Let $g$ be a Two-S-graph for the schema $D$ and the path functional dependency $c(X \to Y)$. As proven in Lem. 4.64, the Two-S-graph $g$ is a root-isomorphic instance of the schema $(D \backslash \Xi) \cup \Xi^{\oplus_D}$ but not an instance of the schema $(D \backslash \Xi) \cup \Xi^{\oplus_D} \cup \{c(X \to Y)\}$. From Lem. 4.68, we know that the operation $\text{merge}_D$ leaves these properties untouched such that $f := \bigcup_{i \in \mathbb{N}} \text{merge}^i_D(g)$ is an instance of the schema $(D \backslash \Xi) \cup \Xi^{\oplus_D}$ but does not satisfy the path functional dependency $c(X \to Y)$.

Due to Def. 4.28 and Lem. 4.12 each derivable class inclusion constraint is satisfied in every extension $\text{merge}^i_D(g)$ and hence $f$.

If object $o$ is generated in the $i$-th iteration, then all onto constraints violated by $o$ are satisfied in the $i+1$-th iteration. This continues throughout further iterations.

Therefore the extension $f$ is an extension of the schema $D$ that does not satisfy the path functional dependency $c(X \to Y)$.

□

## 4.6 Possible Extensions

Certainly, there are numerous possibilities to extend the presented semantic constraints. First of all, the flexibility of F-logic offers the possibility to conceive completely different constraints. Therefore we want to confine our attention to extensions that are based on the constraints we presented so far.

Innate to all presented constraints is the restriction that they are based on scalar attributes. A possible extension is therefore to use set-valued attributes whenever we employed scalar attributes.

Without going into detail, we want to discuss the problems connected with these extensions. We start with allowing even set-valued attributes in the definition of onto constraints. The semantics can even be left unaltered except that we have F-formulae with scalar or set-valued attributes. We surmise that the inference rules for onto constraints and class inclusion constraints remain the same. Matters get more complicated when we bring path functional dependencies into play. But the analysis of the consequences is not within the scope of this work.

The next suggested change deals with onto constraints again. It is possible to replace the simple attribute by a full path function or perhaps with a way description. Again we will not pursue this extension any further.

Possible modifications of path functional dependencies include the use of set-valued attributes in path functions or the use of way descriptions. A similar modification is the use of schema subgraphs instead of path functions. To generalise these ideas, we make the following observation. The principal idea of general functional dependencies is that something uniquely determines something other. These somethings are somehow semantically connected things. The semantic connection can be seen at functional dependencies in the relational data model as attribute values of tuples or sets of path functions as in our approach. The work of Klein and Rasch [KR97] tries to build up a framework to investigate these considerations in more depth. They also give a broad overview over the existing approaches. A crucial point in the generalisation in an object-oriented data model is the identification of objects based on values [AK89, BT98, Kim95, AVdB95, ST93].

# Chapter 5

# From Conceptual to Object-Oriented Models

The ER-model and other conceptual models, which are not object-oriented, have received much interest as starting point for database design transformations. A survey is given by Fahrner and Vossen [FV95]. This survey does not include object-oriented data models as target model. Some work has been done in that direction [NCB92, Get92, HG92, NNJ93, PTCL93, GHC$^+$93, EN94, Bar95, KS95, MGG95, BMP96, ME96, ME98]. A detailed analysis and comparison of these approaches lie beyond the scope of this work. Instead we present the method of Biskup et. al. [BMP96] in this work by means of examples.

For the design of object-oriented databases, entity sets are simply formalised as (entity) classes whose basic types are determined by the pertinent properties of entities. In order to formalise a relationship set, we can always canonically simulate the relational approach [NCB92, Get92, HG92, GHC$^+$93, NNJ93, EN94, BMP96, Rum87].

For every entity we create an object. The properties of the entity are modelled by attributes whose result classes are the classes corresponding to the data types of the properties. Because F-logic is purely object-oriented, values like Ints have to be modelled as objects without internal structure. For every individual relationship we construct an object that is counterpart to the corresponding tuple in the relational approach. Then these objects are understood as instances of a (relationship) class. As objects correspond in this case to tuples, we also need canonical semantic constraints to ensure that they behave as such. These constructs require that the attribute values of a relationship object uniquely represent the relationship, i. e., there is at most one object for any value combination. This kind of constraint is formalised as *key functional dependency* for the class. The construct usually accommodates all attributes of this class.

From the ER-schema in Fig. 1.4, we can derive the schema Assignment. Its components are depicted in Fig. 5.1 and the schema graph in Fig. 5.2. All entity sets are mapped onto corresponding classes, for example the entity set Course onto the class Course. The property title is formalised in the object-oriented data model as the attribute title with result class String. The only relationship set Assignment is mapped onto the (relationship) class Assignment. Its attributes are determined by the entity sets participating in the relationship. Every entity set is formalised as an attribute

$$
\begin{aligned}
\mathrm{CLASS} :=\ & \{\mathsf{String}, \mathsf{Int}, \mathsf{Assignment}, \mathsf{Course}, \mathsf{Teacher}, \mathsf{Assistant}, \mathsf{Date}, \mathsf{Room}, \mathsf{Wing}\} \\
\mathrm{METH} :=\ & \{\mathsf{course}, \mathsf{teacher}, \mathsf{assistant}, \mathsf{room}, \mathsf{wing}, \mathsf{title}, \mathsf{te\_name}, \mathsf{as\_name}, \mathsf{ro\_name}, \\
& \quad \mathsf{date}, \mathsf{year}, \mathsf{month}, \mathsf{day}, \mathsf{size}, \mathsf{address}\} \\
\mathrm{HIER} :=\ & \emptyset \\
\mathrm{SIG} :=\ & \{\mathsf{Course}[\mathsf{title} \Rightarrow \mathsf{String}], \\
& \quad \mathsf{Teacher}[\mathsf{te\_name} \Rightarrow \mathsf{String}], \\
& \quad \mathsf{Assistant}[\mathsf{as\_name} \Rightarrow \mathsf{String}], \\
& \quad \mathsf{Date}[\mathsf{year} \Rightarrow \mathsf{Int}; \mathsf{month} \Rightarrow \mathsf{Int}; \mathsf{day} \Rightarrow \mathsf{Int}], \\
& \quad \mathsf{Room}[\mathsf{size} \Rightarrow \mathsf{Int}; \mathsf{ro\_name} \Rightarrow \mathsf{String}], \\
& \quad \mathsf{Wing}[\mathsf{address} \Rightarrow \mathsf{String}], \\
& \quad \mathsf{Assignment}[\mathsf{course} \Rightarrow \mathsf{Course}; \\
& \qquad\qquad \mathsf{assistant} \Rightarrow \mathsf{Assistant}; \\
& \qquad\qquad \mathsf{date} \Rightarrow \mathsf{Date}; \\
& \qquad\qquad \mathsf{teacher} \Rightarrow \mathsf{Teacher}; \\
& \qquad\qquad \mathsf{room} \Rightarrow \mathsf{Room}; \\
& \qquad\qquad \mathsf{wing} \Rightarrow \mathsf{Wing}]\} \\
\mathrm{SC} :=\ & \mathrm{AX} \cup \\
& \{\mathsf{Assignment}(.\mathsf{course} \rightarrow .\mathsf{assistant}\quad .\mathsf{date}), \\
& \quad \mathsf{Assignment}(.\mathsf{teacher} \rightarrow .\mathsf{room}), \\
& \quad \mathsf{Assignment}(.\mathsf{room} \rightarrow .\mathsf{wing}), \\
& \quad \mathsf{Assignment}(.\mathsf{course}\quad .\mathsf{teacher} \rightarrow .\mathrm{Id})\}
\end{aligned}
$$

Figure 5.1: An object-oriented schema for the "good" example

with the class corresponding to the entity set as result class. Therefore the attribute course with result class Course models the entity set Course. The semantic constraints declared for the class Assignment are of main interest. For every Course there is exactly one Assistant and it takes place at only one Date. The path functional dependency Assignment(.course → .assistant  .date) enforces that this restriction is met. Imagine that in the application at hand a Teacher is assigned a fixed Room, then the path functional dependency Assignment(.teacher → .room) ensures just this requirement. Additionally, the constraint that a Room is situated in only one Wing is reflected in the path functional dependency Assignment(.room → .wing).

The method of Biskup et. al. requires the declaration of two kinds of canonical semantic constraints for the enforcement of:

- *complete* representation of relationships, and

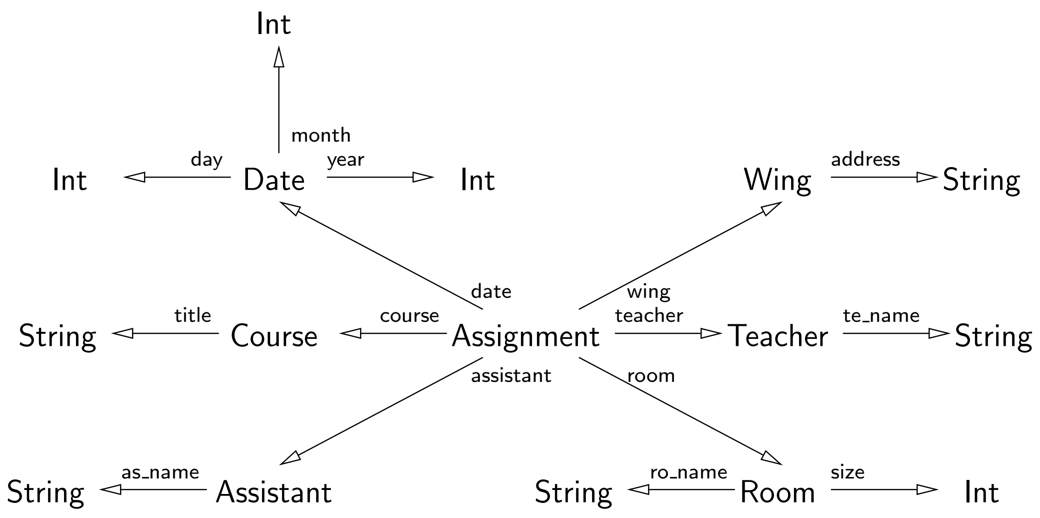


Figure 5.2: The schema graph for the "good" example

- *unique* representation of relationships.

The former ensures that for every (relationship) object all attribute values stemming from the formalisation of the relationship are defined. These constraints are covered by the definedness axioms. The latter, for the unique representation of relationships, are present to ensure that every object of a relationship class behaves as a tuple. These constraints can be expressed in our data model as key functional dependencies. In the example a formalisation of the unique representation of relationships is enforced by the key functional dependency

$$\mathsf{Assignment}(.\mathsf{course}\quad .\mathsf{teacher}\quad .\mathsf{assistant}\quad .\mathsf{date}\quad .\mathsf{room}\quad .\mathsf{wing} \longrightarrow .\mathrm{Id})\ . \tag{5.1}$$

We can reduce the canonical semantic constraint (5.1), which is necessary to make each Assignment-object to simulate a tuple. For the reduction we exploit the path functional dependencies in SC and get the key path functional dependency

$$\mathsf{Assignment}(.\mathsf{course}\quad .\mathsf{teacher} \longrightarrow .\mathrm{Id})\ ,$$

by applying the inference rules path function augmentation and transitivity.

From the ER-schema in Fig. 1.5, we can derive the components of a schema in our F-logic based data model Phone-Admin = $\langle \mathrm{CLASS}|\mathrm{METH}|\mathrm{HIER}|\mathrm{SIG}|\mathrm{SC}\rangle$ as shown in Fig. 5.3. We used the method presented above to generate this schema. Its semantic constraints stem from the formalisation of the restrictions imposed on its instances. The path functional dependency Phone-admin(.fac $\longrightarrow$ .sch) reflects the restriction that a Faculty is affiliated at most to one School. The Faculty is charged with the cost of at most one Phone, hence the path functional dependency Phone-admin(.fac $\longrightarrow$ ph). That a School is accommodated by one Department is formalised by the path functional dependency Phone-admin(.sch $\longrightarrow$ .dep). Finally, the prefix of a Phone number determines the Department. This relationship is captured by the path functional dependency

$$\begin{aligned}
\mathrm{CLASS} &:= \{\mathsf{Phone\text{-}admin}, \mathsf{Faculty}, \mathsf{School}, \mathsf{Phone}, \mathsf{Department}\}\\
\mathrm{METH} &:= \{\mathsf{fac}, \mathsf{sch}, \mathsf{ph}, \mathsf{dep}\}\\
\mathrm{HIER} &:= \emptyset,\\
\mathrm{SIG} &:= \{\mathsf{Phone\text{-}admin}[\mathsf{fac} \Rightarrow \mathsf{Faculty}; \mathsf{sch} \Rightarrow \mathsf{School};\\
&\qquad\qquad\qquad \mathsf{ph} \Rightarrow \mathsf{Phone}; \mathsf{dep} \Rightarrow \mathsf{Department}]\}\\
\mathrm{SC} &:= \mathrm{AX} \cup\\
&\quad \{\mathsf{Phone\text{-}admin}(.\mathsf{fac} \to .\mathsf{sch}),\\
&\quad\; \mathsf{Phone\text{-}admin}(.\mathsf{fac} \to .\mathsf{ph}),\\
&\quad\; \mathsf{Phone\text{-}admin}(.\mathsf{sch} \to .\mathsf{dep}),\\
&\quad\; \mathsf{Phone\text{-}admin}(.\mathsf{ph} \to .\mathsf{dep}),\\
&\quad\; \mathsf{Phone\text{-}admin}(.\mathsf{fac}\quad .\mathsf{sch}\quad .\mathsf{ph}\quad .\mathsf{dep} \to .\mathrm{Id})\}
\end{aligned}$$

Figure 5.3: An object-oriented schema for the “bad” example

$\mathsf{Phone\text{-}admin}(.\mathsf{ph} \to .\mathsf{dep})$. Again the unique representation of relationships is captured as key functional dependency,

$$\mathsf{Phone\text{-}admin}(.\mathsf{fac}\quad .\mathsf{sch}\quad .\mathsf{ph}\quad .\mathsf{dep} \to .\mathrm{Id})\ .$$

# Chapter 6

# Equivalence of Databases

When designing databases, we are often faced with the fact that the same data is structured in various ways. This issue is called *data "relativism"*. It is central to database design. It arises in sundry situations like view construction, view integration, transformations between different data models and interoperable databases. A survey is given in [Hul86, DT93, MIR93]. In our work we concentrate on equivalence of databases given in one data model.

An approach to equivalence of database schemas is to compare their sets of instances. Prominent exponents of this approach are the works of Hull [Hul86] and Biskup et. al. [BR88]. These are the high ends of this kind of work, which are based on the work of Codd [Cod72]. Kohlrausch [Koh96] presents a comparison of these works. The idea underlying these approaches is to consider mappings between sets of instances of two schemas and properties of these mappings. The *symmetric* conception schema equivalence is based on the *asymmetric* conception *dominance*, which captures the idea that one schema has the capacity to contain at least as much information as the other. If two schemas dominate each other, they are *equivalent*.

Unfortunately, the test for dominance, and therefore for equivalence, is not decidable for arbitrary schemas [BR88, MIR94]. Miller et. al. [MIR94] offer an alternative to remedy the dilemma. They develop several tests that serve as sufficient conditions for information capacity dominance or equivalence. Simple schema transformations are introduced. For these transformations the relation with respect to equivalence between input and output schema is known. The composition of simple schema transformations is used to check whether one schema dominates another. The test succeeds if and only if a sequence of simple transformations exists such that one schema can be transformed into the other. Alas, the data model used by Miller et. al., *schema intension graphs (SIG)*, is inheritly *data centric.* In this data model it is impossible to express constraints on the structure of individual entities, i. e., (path) functional dependencies cannot be expressed.

An approach to equivalence of hierarchical data models is the work of Abiteboul and Hull [AH88]. Their data model allows to define types using tuple, set and union constructors. On objects of this data model rewrite operations are defined, which allow to transform objects of one type into objects of a different type. A class of rewrite operations is the class of *simple rewrite operations*. This class is closed under composition. Then transformations on types are presented, called *capacity preserving transformations*. Connected to type transformations are restructuring functions, which can be represented as families of simple rewrite rules. That type transformations are capacity preserving indeed — as the name indicates — is illustrated under the equivalence notion *absolute dominance* [Hul86]. This approach relates the semantic requirement of schema equivalence formalised as dominance to a syntactic property, the presence of simple rewrite operations. However, the data model does not contain object identity.

A different approach to schema equivalence is taken by Qian [Qia96] and Makowsky et. al. [MR96, MR98]. Their works are founded on *signature interpretations* or *translation schemes*, respectively. The central idea is to define the mapping connecting the schemas in a way that it is possible to transform both schemas or at least important schema elements, and instances. Common to both approaches is that a schema defines structure and semantic constraints for its data. It is now possible to translate besides instances arbitrary formulae in the respective data models, and thus semantic constraints. With these means at hand schema equivalence is defined. Unfortunately, the approach of Qian runs into the same undecidability problems as previous works, because the advantage gained due to the ability to transform semantic constraints is neither sufficiently investigated nor exploited. In contrast, Makowsky and Ravve build up an entire framework and accordingly gain a rich harvest from their efforts. They can give syntactic criteria for the test for schema equivalence. Thus they connect the semantic requirement *schema equivalence* with a syntactic property.

## 6.1 Translation Schemes

Although the work of Makowsky and Ravve has been carried out in the context of the relational data model, it can be transferred to our data model as well. This is possible because the work is based on first-order logic. It is grounded on the *atomar* elements of formulae, *positive literals*, likewise it is possible to ground it on the atomar elements of F-logic, namely is-a assertions, object molecules *and* P-molecules, as it was shown in Sect. 2.5 about molecule definitions.

Despite the similarity of using predicate definitions or molecule definitions, there is one fundamental difference between the use of predicate definitions in the relational framework and the use of molecule definitions in F-logic. This difference is rooted in the variance of the conception of a schema. In the relational model a schema can be understood as the declaration of a set of symbols in contrast to the complex declaration of a schema in F-logic. Instead of simply defining a new set of symbols, the values of "built-in" predicates are defined. So we define even the structure of a schema by means of molecule definitions. This is also of advantage, because formulae in F-logic may range over structural aspects such as signatures or the class hierarchy. Having defined schemas by molecule definitions, we have defined the class hierarchy and signatures by molecule definitions. So it becomes possible to translate arbitrary F-formulae. Before we go into detail, we give some definitions.

The transformation pivoting, which lies in the focus of our interest, deals with attributes only. For this reason we restrict our attention to schemas and formulae that involve attributes as only methods.

Definition 6.1 (Simple Structure, Simple Formula, Simple Schema)

- *Let $\mathcal{I}$ be an F-structure. It is called a* simple F-structure *if*
  - $\mathcal{I}_{\rightarrow}$ *is a mapping* $\mathcal{I}_{\rightarrow} : U \mapsto \mathrm{Partial}(U, U)$ *and everywhere else undefined,*
  - *the mapping* $\mathcal{I}_{\twoheadrightarrow}$ *is undefined everywhere,*
  - $\mathcal{I}_{\Rightarrow}$ *is a mapping* $U \mapsto \mathrm{PartialAntiMonotone}(U, \wp_{\uparrow}(U))$ *and everywhere else undefined, and*
  - *the mapping* $\mathcal{I}_{\Rrightarrow}$ *is undefined everywhere.*
- *Let $\alpha$ be a formula. It is called a* simple formula *if in every data expression and every signature expression no arguments occur.*
- *Let $D$ be a schema. It is a* simple schema *if it contains only simple formulae.*

In the sequel we assume that we deal with simple formulae and simple schemas only. In principle it is possible to extend the following definitions to range over ordinary formulae and ordinary schemas though.

A transformation for schemas comes hand in hand with the transformation for their extensions. A translation scheme should accordingly contain definitions for all "built-in" predicates. Additionally, a schema consists of a set of classes and methods. We declare for these "defining formulae" as well. The following definition defines a set of formulae as *pre-translation scheme.* At first we do not couple this set with any schema.

Definition 6.2 (Pre-Translation Scheme)
*A set of formulae*

$$\Phi = \{\phi_{\mathrm{CLASS}}(C), \phi_{\mathrm{METH}}(M), \phi_{::}(C, D), \phi_{\Rightarrow}(M, C, R), \phi_{:}(O, C), \phi_{\rightarrow}(M, O, R)\}$$

*is called a* pre-translation scheme.

Having a schema $D$ and a pre-translation scheme $\Phi$, we define a mapping $\Phi^*$ for schemas. It uses the defining formulae in the pre-translation scheme to deduce a new schema from the input schema. For the elements $\mathrm{CLASS}_{\Phi^*(D)}$, $\mathrm{METH}_{\Phi^*(D)}$, $\mathrm{HIER}_{\Phi^*(D)}$ and $\mathrm{SIG}_{\Phi^*(D)}$ we take the constants, which are plugged into the respective formulae, such that the respective ground formulae are logically implied by the structural part of the schema. The set $\mathrm{SC}_{\Phi^*(D)}$ is simply the set of formulae at least contained by definition.

Definition 6.3
*Let $D$ be a schema, and $\Phi$ be a pre-translation scheme.*

$$\Phi^*(D) := \left\langle \mathrm{CLASS}_{\Phi^*(D)} | \mathrm{METH}_{\Phi^*(D)} | \mathrm{HIER}_{\Phi^*(D)} | \mathrm{SIG}_{\Phi^*(D)} | \mathrm{SC}_{\Phi^*(D)} \right\rangle$$

*is defined as follows:*

- $\text{CLASS}_{\Phi^*(D)} := \{c \mid c \in \mathcal{F}_0$ *and* $D\backslash\text{SC}_D \models \varphi_{\text{CLASS}}(c)\}$, *where*
$$\varphi_{\text{CLASS}}(C) = \phi_{\text{CLASS}}(C) \wedge \neg\phi_{\text{METH}}(C) \ ,$$
- $\text{METH}_{\Phi^*(D)} := \{m \mid m \in \mathcal{F}_0$ *and* $D\backslash\text{SC}_D \models \varphi_{\text{METH}}(m)\}$, *where*
$$\varphi_{\text{METH}}(M) = \phi_{\text{METH}}(M) \wedge \neg\phi_{\text{CLASS}}(M) \ ,$$
- $\text{HIER}_{\Phi^*(D)} := \{c{::}d \mid c, d \in \mathcal{F}_0$ *and* $D\backslash\text{SC}_D \models \varphi_{::}(c,d)\}$, *where*
$$\varphi_{::}(C,D) = \phi_{::}(C,D) \wedge \varphi_{\text{CLASS}}(C) \wedge \varphi_{\text{CLASS}}(D) \ ,$$
- $\text{SIG}_{\Phi^*(D)} := \{c[m \Rightarrow r] \mid m, c, r \in \mathcal{F}_0$ *and* $D\backslash\text{SC}_D \models \varphi_{\Rightarrow}(m,c,r)\}$, *where*
$$\varphi_{\Rightarrow}(M,C,R) = \phi_{\Rightarrow}(M,C,R) \wedge \varphi_{\text{METH}}(M) \wedge \varphi_{\text{CLASS}}(C) \wedge \varphi_{\text{CLASS}}(R) \ ,$$
*and*
- $\text{SC}_{\Phi^*(D)} := \text{AX}$.

In general, $\Phi^*(D)$ is not a schema according to Def. 3.1. We call pre-translation schemes *translation schemes* from a schema if the outcome produces a schema. First of all the syntactic criteria imposed on the sets $\text{CLASS}_{\Phi^*(D)}$, $\text{METH}_{\Phi^*(D)}$, $\text{HIER}_{\Phi^*(D)}$ and $\text{SIG}_{\Phi^*(D)}$ must be satisfied. In addition, we want the formulae to define their output independent from extensions of the input schema. This is impossible for the definition of the class hierarchy, because of the fact that $\forall X\ X{::}X$ is a tautology, which means that the definition of :: depends on the extension. For that reason, we want the definition of the class hierarchy at least to be independent for non-trivial cases, i. e. when there are two different objects on the left-hand side and right-hand side of ::. Since we redefine "built-in" predicates, the defining formulae must form a set of $\mathcal{S}$-definitions. The redefinition is done by first renaming the "built-in" predicates and then defining them again. The renaming is plainly done by translating arbitrary formulae into their relational counterpart.

Definition 6.4 (Translation Scheme)
*Let $\Phi$ be a pre-translation scheme, and $D$ be a schema. $\Phi$ is a* translation scheme from *$D$, if all of the following holds:*

1. $\text{CLASS}_{\Phi^*(D)}$ *is finite and non-empty,*
2. $\text{METH}_{\Phi^*(D)}$ *is finite,*
3. $\text{HIER}_{\Phi^*(D)}$ *is finite,*
4. $\text{SIG}_{\Phi^*(D)}$ *is finite,*
5. *for all constants $c \in \mathcal{F}_0$ and all extensions $f \in \text{ext}(D)$ of $D$:*
$$D\backslash\text{SC}_D \models \varphi_{\text{CLASS}}(c) \textit{ iff } \text{compl}_D(f) \models \varphi_{\text{CLASS}}(c) \ ,$$

6. *for all constants $m \in \mathcal{F}_0$ and all extensions $f \in \mathrm{ext}(D)$ of $D$:*

$$D \backslash \mathrm{SC}_D \models \varphi_{\mathrm{METH}}(m) \textit{ iff } \mathrm{compl}_D(f) \models \varphi_{\mathrm{METH}}(m) \ ,$$

7. *for all constants $c, d \in \mathcal{F}_0$ and all extensions $f \in \mathrm{ext}(D)$ of $D$:*

$$D \backslash \mathrm{SC}_D \models \varphi_{::}(c,d) \textit{ iff } \mathrm{compl}_D(f) \models \varphi_{::}(c,d) \ ,$$

8. *for all constants $m, c, r \in \mathcal{F}_0$ and all extensions $f \in \mathrm{ext}(D)$ of $D$:*

$$D \backslash \mathrm{SC}_D \models \varphi_{\Rightarrow}(m,c,r) \textit{ iff } \mathrm{compl}_D(f) \models \varphi_{\Rightarrow}(m,c,r) \ ,$$

9. *for all extensions $f \in \mathrm{ext}(D)$ of $D$:*

$$\mathrm{compl}_D(f) \models \forall C \forall D(\phi_{::}(C,D) \wedge C \not\overset{\circ}{=} D \Rightarrow \varphi_{\mathrm{CLASS}}(C) \wedge \varphi_{\mathrm{CLASS}}(D))$$

*and*

10. *the set $\Delta^{\Phi}$ with*

$$\begin{aligned}\Delta^{\Phi} := \{ & \forall C \forall D(C{::}D \Leftrightarrow (\phi_{::}(C,D))^r), \\ & \forall M \forall C \forall R(C[M \Rightarrow R] \Leftrightarrow (\varphi_{\Rightarrow}(M,C,R))^r), \\ & \forall M \forall C(C[M \Rightarrow ()] \Leftrightarrow (\exists R\, \varphi_{\Rightarrow}(M,C,R))^r), \\ & \forall O \forall C(O{:}C \Leftrightarrow (\varphi_{:}(O,C))^r), \\ & \forall M \forall O \forall R(O[M \rightarrow R] \Leftrightarrow (\varphi_{\rightarrow}(M,O,R))^r)\} \\ & \cup \\ & \{\forall V_1 \cdots \forall V_{n+1}(f(V_1,\ldots,V_n) \overset{\circ}{=} V_{n+1} \Leftrightarrow p_f(V_1,\ldots,V_{n+1})) \\ & \quad \mid f \in \mathcal{S} \textit{ with arity } n\}\end{aligned}$$

*is a set of $\mathcal{S}$-definitions in the set of sentences as chosen in the proof of Theor. 2.32.*

*We redefine object constructors by giving them their original values.*

*We have to write $\phi_{::}$ in the $\mathcal{S}$-definition of $::$ in the set of $\mathcal{S}$-definitions $\Delta^{\Phi}$, because $\varphi_{::}$ is too restrictive. The "built-in" predicate is reflexive on all elements of the domain including objects.*

It is obvious that a translation scheme from a schema produces a schema as shown in the next lemma.

LEMMA 6.5

*Let $\Phi$ be a translation scheme from a schema $D$. The transformation result $\Phi^*(D)$ is a schema.*

PROOF. The sets $\mathrm{CLASS}_{\Phi^*(D)}$, $\mathrm{METH}_{\Phi^*(D)}$, $\mathrm{HIER}_{\Phi^*(D)}$ and $\mathrm{SIG}_{\Phi^*(D)}$ are finite by the definition of translation schemes. The formulae in the definition of $\Phi^*$ are chosen to reflect the restrictions imposed on schemas and $\mathcal{S}$-definitions.

□

Schemas that are results of the application of a translation scheme are special, because the sets HIER and SIG are nearly logically closed. The set of class declarations HIER violates this because it does not satisfy the tautology $\forall X\, X{::}X$. The set of signature declarations SIG is not logically closed because it does not contain formulae of the form $c[m \Rightarrow (\,)]$ although they are implied by formulae of the form $c[m \Rightarrow r]$.

In general, a schema with

$$\mathrm{HIER}_1 = \{a{::}b, b{::}c\}$$

and a schema with

$$\mathrm{HIER}_2 = \{a{::}a, b{::}b, c{::}c, a{::}b, b{::}c, a{::}c\}$$

are equivalent with respect to their class hierarchy. It is apparent that $\mathrm{HIER}_1 \models\!\!\!\!\!\!=\!\!\!| \mathrm{HIER}_2$. Similar considerations can be made for the set SIG. We extend this idea of equivalence to schemas. In the following definition we see schemas simply as sets of formulae.

Definition 6.6
*Let $D$ and $D'$ be two schemas. $D$ is equivalent to $D'$, written $D \simeq D'$, if*

$$D\backslash\mathrm{SC}_D \models\!\!\!\!\!\!=\!\!\!| D'\backslash\mathrm{SC}_{D'} \quad \textit{and} \quad \mathrm{SC}_D \models\!\!\!\!\!\!=\!\!\!| \mathrm{SC}_{D'} \ .$$

It is easy to prove that a schema $D$ that is equivalent to a schema $D'$ in the sense above can be used interchangeably with it.

We split a schema into its structure giving part and its semantic constraints, because semantic constraints may come in the same form as formulae in the structure giving part.

This property is in particular of interest in the case of schemas that are equivalent to the image of some translation scheme. Therefore we give the following definition, because it enlarges the set of images of a translation scheme substantially.

Definition 6.7
*Let $D$ and $D'$ be two schemas, and $\Phi$ be a translation scheme from $D$. $\Phi$ is a* translation scheme from $D$ to $D'$, *if*

$$\Phi^*(D) \simeq D' \ .$$

Note that $D'$ has only semantic constraints with the following property, $\mathrm{SC}_{D'} \models\!\!\!\!\!\!=\!\!\!| \mathrm{AX}$. Since that is clear, we sometimes give schemas without any semantic constraints not even axioms rather assume that these are automatically added.

The same mechanism that is used to define $\Phi^*$ on a schema can be employed when it comes to extensions. The defining formulae in the translation scheme $\Phi$ are used to deduct the elements pop and ob of an extension. Again there are restrictions imposed on extensions, which have to be obeyed by the formulae.

Definition 6.8
*Let $f$ be an extension of a schema $D$, and $\Phi$ be a translation scheme from $D$.*

$$\Phi^*(f) := \left\langle \mathrm{pop}_{\Phi^*(f)} | \mathrm{ob}_{\Phi^*(f)} \right\rangle$$

*is defined as follows:*

- $\mathrm{pop}_{\Phi^*(f)} := \{o{:}c \mid o, c \in \mathcal{F}_0$ *and* $\mathrm{compl}_D(f) \models \varphi_{:}(o, c)\}$, *where*

$$\varphi_{:}(O, C) = \phi_{:}(O, C) \wedge \neg\varphi_{\mathrm{CLASS}}(O) \wedge \neg\varphi_{\mathrm{METH}}(O) \wedge \varphi_{\mathrm{CLASS}}(C) \ ,$$

*and*

- $\mathrm{ob}_{\Phi^*(f)} := \{o[m \to r] \mid m, o, r \in \mathcal{F}_0$ *and* $\mathrm{compl}_D(f) \models \varphi_{\to}(m, o, r)\}$, *where*

$$\begin{aligned}\varphi_{\to}(M, O, R) = \phi_{\to}(M, O, R) \wedge \varphi_{\mathrm{METH}}(M) \wedge \neg\varphi_{\mathrm{CLASS}}(O) \wedge \neg\varphi_{\mathrm{METH}}(O) \wedge \\ \neg\varphi_{\mathrm{CLASS}}(R) \wedge \neg\varphi_{\mathrm{METH}}(R) \ .\end{aligned}$$

Lemma 6.9
*Let $f$ be an extension of a schema $D$, and $\Phi$ be a translation scheme from $D$. The outcome $\Phi^*(f)$ of applying the mapping $\Phi^*$ to the extension $f$ is an extension of the schema $\Phi^*(D)$.*

Proof. The formulae in the definition of $\Phi$ are chosen to reflect the restrictions imposed on extensions. From that the assertion follows.

□

Makowsky and Ravve [MR96, MR98] define with the help of translation schemes a way to translate formulae. As input they use formulae over the image of the input schema. The output are formulae over the input schema. We can do a similar thing. But as we redefine molecules, we have no such conception of "a formula over a schema". The redefinition is carried out by first renaming the "built-in" predicates and then redefining them based on the renamed predicates. The renaming is simply performed by translating the formulae into their relational counterparts.

Definition 6.10
*Let $\Phi$ be a translation scheme from a schema $D$. The transformation $\Phi^{\#}$ translates formulae into formulae as follows:*

$$\Phi^{\#}(\alpha) := \alpha^{\nabla - r} \ ,$$

*where $\alpha$ is a simple formula and $\Delta := \Delta^{\Phi}$.*

As already mentioned before the sets $\mathrm{HIER}_{\Phi^*(D)}$ and $\mathrm{SIG}_{\Phi^*(D)}$ are nearly logically closed. This property holds also for the components $\mathrm{pop}_{\Phi^*(f)}$ and $\mathrm{ob}_{\Phi^*(f)}$ of a translated extension $\Phi^*(f)$. Lem. 6.11 captures this behaviour formally.

Lemma 6.11
*Let $f$ be an extension of a schema $D$, and $\Phi$ be a translation scheme from $D$.*

1. $c{::}d \in \mathrm{HIER}_{\Phi^*(D)}$ *or* $c \overset{\circ}{=} d \in \mathrm{compl}_D(f)$ *iff* $c{::}d \in \mathrm{compl}_{\Phi^*(D)}(\Phi^*(f))$.
2. $c[m \Rightarrow r] \in \mathrm{SIG}_{\Phi^*(D)}$ *iff* $c[m \Rightarrow r] \in \mathrm{compl}_{\Phi^*(D)}(\Phi^*(f))$.
3. $o{:}c \in \mathrm{pop}_{\Phi^*(f)}$ *iff* $o{:}c \in \mathrm{compl}_{\Phi^*(D)}(\Phi^*(f))$.

4. $o[m \rightarrow r] \in \mathrm{ob}_{\Phi^*(f)}$ *iff* $o[m \rightarrow r] \in \mathrm{compl}_{\Phi^*(D)}(\Phi^*(f))$.

PROOF. The definitions of $\mathrm{HIER}_{\Phi^*(D)}$, $\mathrm{SIG}_{\Phi^*(D)}$, $\mathrm{pop}_{\Phi^*(f)}$ and $\mathrm{ob}_{\Phi^*(f)}$ are based on formulae that are $\mathcal{S}$-definitions. These are defined in a way that they are "logically closed" as $\mathrm{compl}_{\Phi^*(D)}(\Phi^*(f))$. □

Because of Theor. 2.29, Theor. 2.33 and Def. 6.10 a formula $\alpha^{\nabla-r}(V_1,\ldots,V_n)$ exists for every formula $\alpha(V_1,\ldots,V_n)$ such that for every simple structure $\mathcal{I}$ and $u_1,\ldots,u_n \in \mathcal{I}(U)$:

$$\mathcal{I} \models_\nu \alpha^{\nabla-r} \text{ iff } \mathcal{I}^r \models_\nu \alpha^\nabla \text{ iff } \mathcal{I}^{r\Delta} \models_\nu \alpha$$

where $\nu(V_i) = u_i$.

Moreover $\mathrm{compl}_D(f)$ is the smallest H-model of $D\backslash\mathrm{SC}_D \cup f$ and a simple structure, and so

$$\mathrm{compl}_D(f) \models \alpha^{\nabla-r} \text{ iff } \mathrm{compl}_D(f)^r \models \alpha^\nabla \text{ iff } \mathrm{compl}_D(f)^{r\Delta} \models \alpha\ .$$

A schema $D$ and an instance $f$ span an H-model namely $\mathrm{compl}_D(f)$. We are interested in the H-model $\mathrm{compl}_{\Phi^*(D)}(\Phi^*(f))$ spanned by $\Phi^*(D)$ and $\Phi^*(f)$. What is its relation to $\mathrm{compl}_D(f)^{r\Delta}$? It turns out that $\mathrm{compl}_{\Phi^*(D)}(\Phi^*(f))$ is the restriction of $\mathrm{compl}_D(f)^{r\Delta}$ to the set $\mathcal{F} \cup \{\overset{\circ}{=}\}$ of symbols.

LEMMA 6.12
*Let $f$ be an extension of a schema $D$, and $\Phi$ be a translation scheme from $D$.*

1. $\mathrm{compl}_D(f)^{r\Delta} \models c{::}d$ *iff* $\mathrm{compl}_{\Phi^*(D)}(\Phi^*(f)) \models c{::}d$.
2. $\mathrm{compl}_D(f)^{r\Delta} \models c[m \Rightarrow r]$ *iff* $\mathrm{compl}_{\Phi^*(D)}(\Phi^*(f)) \models c[m \Rightarrow r]$.
3. $\mathrm{compl}_D(f)^{r\Delta} \models c[m \Rightarrow (\,)]$ *iff* $\mathrm{compl}_{\Phi^*(D)}(\Phi^*(f)) \models c[m \Rightarrow (\,)]$.
4. $\mathrm{compl}_D(f)^{r\Delta} \models o : c$ *iff* $\mathrm{compl}_{\Phi^*(D)}(\Phi^*(f)) \models o : c$.
5. $\mathrm{compl}_D(f)^{r\Delta} \models o[m \rightarrow r]$ *iff* $\mathrm{compl}_{\Phi^*(D)}(\Phi^*(f)) \models o[m \rightarrow r]$.
6. $\mathrm{compl}_D(f)^{r\Delta} \models a \overset{\circ}{=} b$ *iff* $\mathrm{compl}_{\Phi^*(D)}(\Phi^*(f)) \models a \overset{\circ}{=} b$.

PROOF. In this proof we make not such a big fuss about translating the formulae as we do in the proof of Theor. 2.29. We have only ground formulae in this case. So the translation boils down to simple replacements.

FOR 1.

$\mathrm{compl}_D(f)^{r\Delta} \models c{::}d$ iff $\mathrm{compl}_D(f)^r \models (c{::}d)^\nabla$ by Theor. 2.29
iff $\mathrm{compl}_D(f)^r \models \phi_{::}(c,d)^r$
by Def. $\alpha^\nabla$ (Proof of Theor. 2.29)
iff $\mathrm{compl}_D(f) \models \phi_{::}(c,d)$ by Theor. 2.32
iff $D\backslash\mathrm{SC}_D \models \varphi_{::}(c,d)$ or $\mathrm{compl}_D(f) \models c \overset{\circ}{=} d$
by Def. 6.4 (7, 9, 10)
iff $c{::}d \in \mathrm{HIER}_{\Phi^*(D)}$ or $\mathrm{compl}_D(f) \models c \overset{\circ}{=} d$
by Def. 6.3
iff $\mathrm{compl}_{\Phi^*(D)}(\Phi^*(f)) \models c{::}d$ by Lem. 6.12.

For 2.

$$\begin{aligned}
\mathrm{compl}_D(f)^{r\Delta} \models c[m \Rightarrow r] \quad & \text{iff} \quad \mathrm{compl}_D(f)^r \models (c[m \Rightarrow r])^\nabla \\
& \text{iff} \quad \mathrm{compl}_D(f)^r \models \varphi_\Rightarrow(m,c,r)^r \\
& \text{iff} \quad \mathrm{compl}_D(f) \models \varphi_\Rightarrow(m,c,r) \\
& \text{iff} \quad D\backslash \mathrm{SC}_D \models \varphi_\Rightarrow(m,c,r) && \text{by Def. 6.4 (8)} \\
& \text{iff} \quad c[m \Rightarrow r] \in \mathrm{SIG}_{\Phi^*(D)} \\
& \text{iff} \quad \mathrm{compl}_{\Phi^*(D)}(\Phi^*(f)) \models c[m \Rightarrow r] \\
& \quad .
\end{aligned}$$

For 3. Because of its definition $\mathrm{compl}_D(f)^{r\Delta} \models c[m \Rightarrow (\,)]$ if there exists $r \in \mathcal{F}_0$ such that $\mathrm{compl}_D(f)^{r\Delta} \models c[m \Rightarrow r]$. According to 2 we know $\mathrm{compl}_{\Phi^*(D)}(\Phi^*(f)) \models c[m \Rightarrow r]$, hence $\mathrm{compl}_{\Phi^*(D)}(\Phi^*(f)) \models c[m \Rightarrow (\,)]$. The same line of argumentation can be followed when we show the reverse direction of the proof.

For 4.

$$\begin{aligned}
\mathrm{compl}_D(f)^{r\Delta} \models o{:}c \quad & \text{iff} \quad \mathrm{compl}_D(f)^r \models (o{:}c)^\nabla \\
& \text{iff} \quad \mathrm{compl}_D(f)^r \models \varphi_{:}(o,c)^r \\
& \text{iff} \quad \mathrm{compl}_D(f) \models \varphi_{:}(o,c) \\
& \text{iff} \quad o{:}c \in \mathrm{pop}_{\Phi^*(f)} && \text{by Def. 6.8} \\
& \text{iff} \quad \mathrm{compl}_{\Phi^*(D)}(\Phi^*(f)) \models o{:}c \quad .
\end{aligned}$$

For 5.

$$\begin{aligned}
\mathrm{compl}_D(f)^{r\Delta} \models o[m \rightarrow r] \quad & \text{iff} \quad \mathrm{compl}_D(f)^r \models (o[m \rightarrow r])^\nabla \\
& \text{iff} \quad \mathrm{compl}_D(f)^r \models \varphi_\rightarrow(m,o,r)^r \\
& \text{iff} \quad \mathrm{compl}_D(f) \models \varphi_\rightarrow(m,o,r) \\
& \text{iff} \quad o[m \rightarrow r] \in \mathrm{ob}_{\Phi^*(f)} && \text{by Def. 6.8} \\
& \text{iff} \quad \mathrm{compl}_{\Phi^*(D)}(\Phi^*(f)) \models o[m \rightarrow r] \quad .
\end{aligned}$$

For 6.

$$\begin{aligned}
\mathrm{compl}_D(f)^{r\Delta} \models a \overset{\circ}{=} b \quad & \text{iff} \quad \mathrm{compl}_D(f)^r \models (a \overset{\circ}{=} b)^\nabla \\
& \text{iff} \quad \mathrm{compl}_D(f)^r \models a \overset{\circ}{=} b \\
& \text{iff} \quad \mathrm{compl}_D(f) \models a \overset{\circ}{=} b \\
& \text{iff} \quad \mathrm{compl}_{\Phi^*(D)}(\Phi^*(f)) \models a \overset{\circ}{=} b \quad .
\end{aligned}$$

□

We summarise the results of the preceding pages.

Theorem 6.13
*Let $\Phi$ be a translation scheme from a schema $D$, and $\alpha(V_1, \ldots, V_n)$ be a formula. Then for all extensions $f \in \mathrm{ext}(D)$ of $D$ and all $u_1, \ldots, u_n \in \mathrm{compl}_D(f)(U)$:*

$$\mathrm{compl}_D(f) \models_\nu \Phi^\#(\alpha) \textit{ iff } \mathrm{compl}_{\Phi^*(D)}(\Phi^*(f)) \models_\nu \alpha \ ,$$

*where $\nu(V_i) = u_i$.*

Proof. By Theor. 2.33 and Theor. 2.29 we have

$$\begin{array}{rll} \mathrm{compl}_D(f) \models_\nu \alpha^{\nabla - r} & \text{iff} & \mathrm{compl}_D(f)^r \models_\nu \alpha^\nabla \\ & \text{iff} & \mathrm{compl}_D(f)^{r\Delta} \models_\nu \alpha \ . \end{array}$$

By induction on the structure of $\alpha$, we can show that

$$\mathrm{compl}_D(f)^{r\Delta} \models_\nu \alpha \text{ iff } \mathrm{compl}_{\Phi^*(D)}(\Phi^*(f)) \models_\nu \alpha \ .$$

For atomar formulae we make use of Lem. 6.12. The rest is then straightforward. □

In [MR98] this is called the *fundamental property of translation schemes*. It shows that the mappings $\Phi^*$ and $\Phi^\#$ are dual to each other. By its help we are enabled to relate semantic requirements desirable for database design to syntactic properties [Bis95a].

Definition 6.14 (Weak Reduction and Reduction)
*Let $\Phi$ be a translation scheme from $D$, $K_R \subset \mathrm{ext}(D)$ be a set of extensions of $D$, and $K_S \subset \mathrm{ext}(\Phi^*(D))$ be a set of extensions of $\Phi^*(D)$. The translation scheme $\Phi$ is a* weak reduction (reduction) from $K_R$ to $K_S$ *if for every extension $f \in K_R$ implies that (iff) $\Phi^*(f) \in K_S$.*

In this work we are interested in families $K_R$ and $K_S$ such that $K_R = \mathrm{sat}(D)$ and $K_S = \mathrm{sat}(D')$ for a translation scheme $\Phi$ from $D$ to $D' \backslash \mathrm{SC}_{D'}$.

The definition of a translation scheme does not touch semantic constraints of the original schema. Theorem 6.15 shows the relation between the semantic constraints of the original schema and an arbitrary set of semantic constraints for the output schema with respect to properties underlying translation schemes, namely weak reduction and reduction.

Theorem 6.15
*Let $\Phi$ be a translation scheme from $D$, and* SC *be a set of sentences.*

1. *The translation scheme $\Phi$ is a weak reduction from* $\mathrm{sat}(D)$ *to* $\mathrm{sat}(\Phi^*(D) \cup \mathrm{SC})$ *iff* $\mathrm{sat}(D) \subset \mathrm{sat}(D \backslash \mathrm{SC}_D \cup \Phi^\#(\mathrm{AX} \cup \mathrm{SC}))$.[1]

2. *The translation scheme $\Phi$ is a reduction from* $\mathrm{sat}(D)$ *to* $\mathrm{sat}(\Phi^*(D) \cup \mathrm{SC})$ *iff* $\mathrm{sat}(D) = \mathrm{sat}(D \backslash \mathrm{SC}_D \cup \Phi^\#(\mathrm{AX} \cup \mathrm{SC}))$.

Proof for 1. $\Longrightarrow$:

$$\begin{array}{rlll} f \in \mathrm{sat}(D) & \text{then} & \Phi^*(f) \in \mathrm{sat}(\Phi^*(D) \cup \mathrm{SC}) & (\Phi \text{ weak reduction}) \\ & \text{iff} & \mathrm{compl}_{\Phi^*(D)}(\Phi^*(f)) \models \mathrm{AX} \cup \mathrm{SC} & \\ & \text{iff} & \mathrm{compl}_D(f) \models \Phi^\#(\mathrm{AX} \cup \mathrm{SC}) & (6.13) \\ & \text{iff} & f \in \mathrm{sat}(D \backslash \mathrm{SC}_D \cup \Phi^\#(\mathrm{AX} \cup \mathrm{SC})) \ . & \end{array}$$

[1] Originally, we define the set of instances sat only for schemas. The "schema" $D \backslash \mathrm{SC}_D \cup \Phi^\#(\mathrm{AX} \cup \mathrm{SC})$ might not be a proper schema according to Def. 3.1, because it is not guaranteed that $\mathrm{AX} \subset \Phi^\#(\mathrm{AX} \cup \mathrm{SC})$ or at least $\Phi^\#(\mathrm{AX} \cup \mathrm{SC}) \models \mathrm{AX}$. However, the set sat of instances is well defined for $D \backslash \mathrm{SC}_D \cup \Phi^\#(\mathrm{AX} \cup \mathrm{SC})$. Therefore we use it even in this context.

$\Longleftarrow$:

$$\begin{array}{lll} f \in \mathrm{sat}(D) & \text{then} & f \in \mathrm{sat}(D\backslash \mathrm{SC}_D \cup \Phi^{\#}(\mathrm{AX} \cup \mathrm{SC})) \quad \text{(by hypothesis)} \\ & \text{iff} & \mathrm{compl}_D(f) \models \Phi^{\#}(\mathrm{AX} \cup \mathrm{SC}) \\ & \text{iff} & \mathrm{compl}_{\Phi^*(D)}(\Phi^*(f)) \models \mathrm{AX} \cup \mathrm{SC} \quad (6.13) \\ & \text{iff} & \Phi^*(f) \in \mathrm{sat}(\Phi^*(D) \cup \mathrm{SC}) \ . \end{array}$$

Proof for 2. $\Longrightarrow$:

$$\begin{array}{lll} f \in \mathrm{sat}(D) & \text{iff} & \Phi^*(f) \in \mathrm{sat}(\Phi^*(D) \cup \mathrm{SC}) \quad (\Phi \text{ reduction}) \\ & \text{iff} & \mathrm{compl}_{\Phi^*(D)}(\Phi^*(f)) \models \mathrm{AX} \cup \mathrm{SC} \\ & \text{iff} & \mathrm{compl}_D(f) \models \Phi^{\#}(\mathrm{AX} \cup \mathrm{SC}) \quad (6.13) \\ & \text{iff} & f \in sat(D\backslash \mathrm{SC}_D \cup \Phi^{\#}(\mathrm{AX} \cup \mathrm{SC})) \end{array}$$

$\Longleftarrow$:

$$\begin{array}{lll} f \in \mathrm{sat}(D) & \text{iff} & f \in \mathrm{sat}(D\backslash \mathrm{SC}_D \cup \Phi^{\#}(\mathrm{AX} \cup \mathrm{SC})) \quad \text{(by hypothesis)} \\ & \text{iff} & \mathrm{compl}_D(f) \models \Phi^{\#}(\mathrm{AX} \cup \mathrm{SC}) \\ & \text{iff} & \mathrm{compl}_{\Phi^*(D)}(\Phi^*(f)) \models \mathrm{AX} \cup \mathrm{SC} \\ & \text{iff} & \Phi^*(f) \in \mathrm{sat}(\Phi^*(D) \cup \mathrm{SC}) \end{array}$$

□

In [MR98] it is shown that this result is equivalent to syntactic properties. In this case the syntactic properties deal with the implication of semantic constraints. In our framework we cannot transfer their result directly. The difficulty stems from the difference that we redefine "built-in" predicates whereas they solely define new predicates. Lemma 4.1 and Exam. 4.2 shed more light on this issue.

Now we put translation schemes to work.

When we deal with database transformations, it is one objective that we can recover the original instances from the transformed instances. In other words we need the left inverse for the instance transformation. This means the database transformation must be *information preserving*. Information preservation is also called *losslessness*, in particular, in literature about relational normal forms[Ull88, Dat94, EN94, AHV95]. There it appears in the context of decomposing relational schemas to obtain relational normal forms.

Definition 6.16 (Information Preservation)
*Let $D$ be a schema, $\Phi$ be a translation scheme from $D$, and $\Psi$ be a translation scheme from $\Phi^*(D)$ to $D\backslash\mathrm{SC}_D$. The translation scheme $\Phi$ is* information preserving on schema *$D$ with left inverse $\Psi$, if for every instance $f \in \mathrm{sat}(D)$ of $D$:*

$$\mathrm{compl}_D(f) = \mathrm{compl}_{\Psi^*(\Phi^*(D))}(\Psi^*(\Phi^*(f))) \ .$$

If a translation scheme $\Phi$ from schema $S$ to schema $T\backslash\mathrm{SC}_T$ is information preserving on $S$ and a weak reduction from $\mathrm{sat}(S)$ to $\mathrm{sat}(T)$, we say in the notation of Hull [Hul86] *$T$ dominates $S$*. It is known [ABM80] that this notion is equivalent to query-dominance [Cod72].

When we try to draw the connection between this semantic requirement and syntactic properties, we discover the following. The syntactic property is based on the logical implication.

THEOREM 6.17
*Let $D$ be a schema, $\Phi$ be a translation scheme from $D$, and $\Psi$ be a translation scheme from $\Phi^*(D)$ to $D\backslash\mathrm{SC}_D$. The translation scheme $\Phi$ is information preserving on schema $D$ with left inverse $\Psi$ if for every formula $\alpha$ we have $\mathrm{SC}_D \models (\alpha \Leftrightarrow \Phi^{\#}(\Psi^{\#}(\alpha)))$.*

PROOF. Let $\delta$ be an element of $\mathrm{compl}_D(f)$ for an instance $f \in \mathrm{sat}(D)$ of $D$. Because of $f \in \mathrm{sat}(D)$, we know that $\mathrm{compl}_D(f)$ is an model of $\mathrm{SC}_D$. We take $\mathrm{compl}_D(f)$ now in its rôle as H-model of $\delta$, $\mathrm{compl}_D(f) \models \delta$. Then under the assumption $\mathrm{SC}_D \models (\varphi \Leftrightarrow \Phi^{\#}(\Psi^{\#}(\varphi)))$ we get $\mathrm{compl}_D(f) \models \Phi^{\#}(\Psi^{\#}(\delta))$. Using Theor. 6.13 twice, once for $\Phi$ and once for $\Psi$, we get $\mathrm{compl}_{\Psi^*(\Phi^*(D))}(\Psi^*(\Phi^*(f))) \models \delta$, and therefore $\mathrm{compl}_D(f) \subset \mathrm{compl}_{\Psi^*(\Phi^*(D))}(\Psi^*(\Phi^*(f)))$. The proof of the inclusion $\mathrm{compl}_D(f) \supset \mathrm{compl}_{\Psi^*(\Phi^*(D))}(\Psi^*(\Phi^*(f)))$ follows the same lines of argumentation. □

Unfortunately, this theorem is not as strong as the corresponding one in [MR98]. In their framework the equivalence holds. The reason is that the notion of a schema in their framework only gives a set of symbols. Therefore, their extensions are logical structures and vice versa, that is every $\mathcal{S}$-structure where $\mathcal{S}$ is a set of symbols given by a schema declaration is an extension. Our notion of a schema is different. Therefore, we have to speak about the completion of an extension under a schema. Though every completion of an extension under a schema is an H-structure, not every F-structure is a completion of some extension under some schema. Even, when it satisfies the semantic constraints. We conjecture that exactly that is possible when we take a different definition of a schema. Essentially the definition is the same. The only difference is that all restrictions are expressed by semantic constraints. Although this approach seems promising, it lies beyond the scope of this work.

Besides the desire to recover every original database instance, we want that the transformed schema does not allow to store inconsistent data from the point of view of the original schema. In order to achieve that goal, the semantic constraints of the original schema must be preserved in the transformed schema. Makowsky and Ravve [MR98] discuss several alternatives to define *dependency preservation* in their framework. Because of the similarity of our work to their work we will not repeat this discussion. Instead we decide to take what they call the most natural choice for the definition of dependency preservation (Option (B)).

DEFINITION 6.18 (DEPENDENCY PRESERVATION)
*Let $T$ and $S$ be two schemas, and $\Psi$ be a translation scheme from $T$ to $S\backslash\mathrm{SC}_S$. The translation scheme $\Psi$ is* dependency preserving on $S$ *if* $\mathrm{SC}_T \models \Psi^{\#}(\mathrm{SC}_S)$.

In the definition above we deviate slightly from the original definition. The rationale for that is that their definition is given in a setting where three alternatives are discussed. This setting comes with an overhead, which we discard in our context.

By aid of these two definitions we define *translation-refinement* and *translation-equivalence*. Note that the hypotheses information preservation and weak reduction correspond to the notion of dominance as defined by Hull [Hul86].

Definition 6.19 (Translation Refinement and Equivalence)
*Let $S$ and $T$ be two schemas.*

- *We say that $T$ is a* translation refinement *of $S$ if there is a translation scheme $\Phi$ from $S$ to $T\backslash \mathrm{SC}_T$ that is a weak reduction from* $\mathrm{sat}(S)$ *to* $\mathrm{sat}(T)$, *information preserving on $S$ with left inverse $\Psi$, and $\Psi$ is dependency preserving on $S$.*

- *We say that schemas $S$ and $T$ are* translation equivalent *if $S$ is a translation refinement of $T$ and $T$ is a translation refinement of $S$.*

Before we embark on a comparison of these definitions with the definitions of Qian, we point out that a translation refinement has an impact on the accompanying translation schemes.

Lemma 6.20
*Let $T$ and $S$ be two schemas, and $\Psi$ be a translation scheme from $T$ to $S\backslash \mathrm{SC}_S$ that is dependency preserving on $S$. Then the translation scheme $\Psi$ is a weak reduction from* $\mathrm{sat}(T)$ *to* $\mathrm{sat}(S)$.

Proof. By the hypothesis we have $\mathrm{SC}_T \models \Psi^{\#}(\mathrm{SC}_S)$. This implies $\mathrm{sat}(T) \subset \mathrm{sat}(T\backslash \mathrm{SC}_T \cup \Psi^{\#}(\mathrm{SC}_S))$ by Lem. 4.1. This inclusion entails the assertion by Theor. 6.15. □

With the lemma above we can then prove that a translation scheme $\Phi$ as in Def. 6.19 is a reduction from $\mathrm{sat}(S)$ to $\mathrm{sat}(T)$.

Theorem 6.21
*Let $S$ and $T$ be two schemas such that $T$ is a translation refinement of $S$, and $\Phi$ and $\Psi$ be the corresponding translation schemes. Then the translation scheme $\Phi$ is a reduction from* $\mathrm{sat}(S)$ *to* $\mathrm{sat}(T)$.

Proof. Because of the hypothesis, translation scheme $\Phi$ is a weak reduction from $\mathrm{sat}(S)$ to $\mathrm{sat}(T)$. And so it is sufficient to show for all extensions $f \in \mathrm{ext}(S)$ of $S$ if $\Phi^*(f) \in \mathrm{sat}(T)$, then $f \in \mathrm{sat}(S)$.

If $\Phi^*(f) \in \mathrm{sat}(T)$ for an extension $f \in \mathrm{ext}(S)$ of $S$, then $\Psi^*(\Phi^*(f)) \in \mathrm{sat}(S)$ because $\Psi$ is a weak reduction from $\mathrm{sat}(T)$ to $\mathrm{sat}(S)$. By the hypothesis that the translation scheme $\Phi$ is information preserving with left inverse $\Psi$, we know that $\mathrm{compl}_D(f) = \mathrm{compl}_{\Psi^*(\Phi^*(D))}(\Psi^*(\Phi^*(f)))$, hence $f \in \mathrm{sat}(S)$.

□

## 6.2 Signature Interpretations

We compare the definitions above with the definitions of Qian [Qia96]. His notion of a *signature interpretation* $\sigma$ corresponds to an induced mapping $\Phi^{\#}$. From this he derives an induced mapping $M_\sigma$ that corresponds to $\Phi^*$. So this correspondence renders it possible to compare the definitions in our framework. We do so by restating his definitions. We begin with his definition of a *constraint-preserving transformation.*

Definition 6.22 (Constraint-Preserving Transformation)
*Let $S$ and $T$ be two schemas, and $\Phi$ be a translation scheme from $S$ to $T\backslash\mathrm{SC}_T$. We say that $\Phi$ is a* constraint-preserving transformation from $S$ to $T$ *if*

$$\mathrm{SC}_S \models \Phi^{\#}(\mathrm{SC}_T) \ .$$

We observe that the preceding definition is identical to our Definition "Dependency Preservation". (Notice that we have to swap the schemas.) This in turn means that $\Phi$ is a weak reduction from $\mathrm{sat}(S)$ to $\mathrm{sat}(T)$ by Lem. 6.20.

The next definition in [Qia96] is that of an *instance-preserving transformation.*

Definition 6.23 (Instance-Preserving Transformation)
*Let $S$ and $T$ be two schemas, and $\Phi$ be a translation scheme from the schema $S$ to the schema $T\backslash\mathrm{SC}_T$. We say that $\Phi$ is an* instance-preserving transformation from $S$ to $T$ *if, for every instance $f_T \in \mathrm{sat}(T)$ of the schema $T$, there is an instance $f_S \in \mathrm{sat}(S)$ of the schema $S$ such that* $\mathrm{compl}_T(\Phi^*(f_S)) = \mathrm{compl}_T(f_T)$.

This definition finds no direct counterpart in our framework. But in combination with Def. 6.22 (Constraint-Preserving Transformation) Qian defines *information-preserving transformations.* For these we are able to give the correspondence in our framework.

Definition 6.24 (Information-Preserving Transformation)
*Let $S$ and $T$ be two schemas, and $\Phi$ be a translation scheme. We say that $\Phi$ is an* information-preserving transformation from $S$ to $T$ *if it is both a constraint-preserving and instance-preserving transformation from $S$ to $T$.*

Note that although the definitions of dependency preservation and constraint-preserving transformations are identical, Qian uses a constraint-preserving transformation from $S$ to $T$ in the preceding definition and we use a dependency preserving translation scheme in the reverse direction, namely from $T$ to $S$ in the definition of translation refinements.

If we have a translation scheme that is an information-preserving transformation, then the translation scheme is a reduction.

Lemma 6.25
*Let $S$ and $T$ be two schemas, and $\Phi$ be an information-preserving transformation from $S$ to $T$. The translation scheme $\Phi$ is a reduction from* $\mathrm{sat}(S)$ *to* $\mathrm{sat}(T)$.

Proof. As indicated before $\Phi$ being a constraint-preserving transformation from $S$ to $T$ implies that $\Phi$ is a weak reduction from $\mathrm{sat}(S)$ to $\mathrm{sat}(T)$. It is even a reduction from $\mathrm{sat}(S)$ to $\mathrm{sat}(T)$, because it is an instance-preserving transformation from $S$ to $T$. $\square$

## 6.3 View Instance Support

We give now the definition of *view instance support* from Biskup [Bis95a]. His stance is that all aspects of the original schema have to be supported by the transformed schema as a view. The definition we use here deviates slightly from the original definition, because we work on saturated instances. In this context, we use on the one hand queries over a schema, whose semantics we comprehend not only in the narrow sense of Def. 3.15, on the other hand the result of a query should deliver only sets conforming to extensions.

Definition 6.26 (View Instance Support)

- *A schema $S$ provides* view instance support *for a schema $T$ if there exists a query $Q$ over $S$ such that*

$$\{\mathrm{compl}_T(f_T) \mid f_T \in \mathrm{sat}(T)\} \subset \{\mathrm{compl}_T(\mathrm{eval}(Q, f_S)) \mid f_S \in \mathrm{sat}(S)\} \ .$$

- *If we have equality, the view instance support is called* faithful*.*

- *If additionally the supporting query is injective on the saturated elements of* $\mathrm{sat}(S)$*, i. e., if* $\mathrm{eval}(Q, f) = \mathrm{eval}(Q, f')$*, then* $\mathrm{compl}_S(f) = \mathrm{compl}_S(f')$*, the view instance support is called* unique*.*

A property of view instance support that we need in the following sections is that view instance support is transitive.

Lemma 6.27 (Transitivity of (Faithful, Unique) View Instance Support)
*Let $S$, $T$, $V$ be schemas such that the schema $T$ provides (faithful, unique) view instance support for the schema $S$ with supporting query $Q_{T\to S}$ and such that the schema $V$ provides (faithful, unique) view instance support for the schema $T$ with supporting query $Q_{V\to T}$. Then the schema $V$ provides (faithful, unique) view instance support for the schema $S$ with supporting query $Q_{T\to S} \circ Q_{V\to T}$.*

Proof. We have to prove the subset relation

$$\begin{aligned}\{\mathrm{compl}_S(f_S) \mid f_S \in \mathrm{sat}(S)\} \subset \\ \{\mathrm{compl}_S(\mathrm{eval}_T(Q_{T\to S}, \mathrm{eval}_V(Q_{V\to T}, f_V))) \mid f_V \in \mathrm{sat}(V)\} \ . \quad (6.1)\end{aligned}$$

Now be $f_S \in \mathrm{sat}(S)$ an instance of the schema $S$, then due to the view instance support of the schema $T$ for the schema $S$, there exists an instance $f_T \in \mathrm{sat}(T)$ of the schema $T$, such that

$$\mathrm{compl}_S(f_S) = \mathrm{compl}_S(\mathrm{eval}_T(Q_{T\to S}, f_T)) \quad (6.2)$$

holds. Likewise there exists an instance $f_V \in \mathrm{sat}(V)$ of the schema $V$ such that

$$\mathrm{compl}_T(f_T) = \mathrm{compl}_T(\mathrm{eval}_V(Q_{V\to T}, f_V))$$

holds. By Lem. 3.18, this equality entails

$$\mathrm{eval}_T(Q_{T\to S}, f_T) = \mathrm{eval}_T(Q_{T\to S}, \mathrm{eval}_V(Q_{V\to T}, f_V)) \ .$$

Replacing $\mathrm{eval}_T(Q_{T\to S}, f_T)$ in (6.2) with $\mathrm{eval}_T(Q_{T\to S}, \mathrm{eval}_V(Q_{V\to T}, f_V))$, we get

$$\mathrm{compl}_S(f_S) = \mathrm{compl}_S(\mathrm{eval}_T(Q_{T\to S}, \mathrm{eval}_V(Q_{V\to T}, f_V))) \ .$$

Hence, the schema $V$ provides view instance support for the schema $S$ with supporting query $Q_{T\to S} \circ Q_{V\to T}$.

Analogous, we can show that the schema $V$ provides faithful view instance support for the schema $S$ with supporting query $Q_{T\to S} \circ Q_{V\to T}$.

Since the concatenation of two injective functions is an injective function again, the schema $V$ provides unique view instance support for the schema $S$ with supporting query $Q_{T\to S} \circ Q_{V\to T}$. □

## 6.4 Comparison

We want to compare all approaches. In order to do so, we observe that the mapping $\Phi^*$ induced by a translation scheme $\Phi$ from a schema $S$ to the schema $T \simeq \Phi^*(S)$ can be taken as a query over $S$. So in that respect we can abstract from the actual definitions and see them merely as abstract mappings on instances. For the comparison we identify the following properties of these mappings and relate them to already introduced properties.

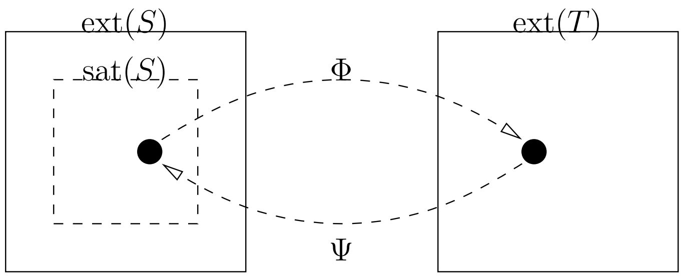
We start off with translation schemes. Their properties are based on the properties information preservation and dependency preservation. In Figure 6.1(a) we indicate that every instance $f_S \in \mathrm{sat}(S)$ of schema $S$ can be mapped onto an extension $\Phi^*(f_S) \in \mathrm{ext}(T)$ of schema $T$. The left inverse $\Psi$ maps $\Phi^*(f_S)$ back onto $f_S$. Actually $\Psi^*(\Phi^*(f_S))$ is in general not equal to $f_S$. Only $\mathrm{compl}_{\Psi^*(\Phi^*(D))}(\Psi^*(\Phi^*(f_S))) = \mathrm{compl}_D(f_S)$ is guaranteed. Nevertheless, we continue with this sloppiness.

The property that $\Phi$ is a weak reduction implies that every image $\Phi^*(f_S)$ of an instance $f_S \in \mathrm{sat}(S)$ of $S$ is an instance of $T$, $\Phi^*(f_S) \in \mathrm{sat}(T)$. Figure 6.1(b) shows this situation. The rectangle drawn with dashed lines on the left side of the diagram represents $\mathrm{sat}(S)$. This rectangle is mapped by $\Phi$ or better its induced mapping $\Phi^*$ onto the rectangle with dashed boundaries on the right side. It lies completely in the dotted rectangle. The latter rectangle depicts $\mathrm{sat}(T)$.

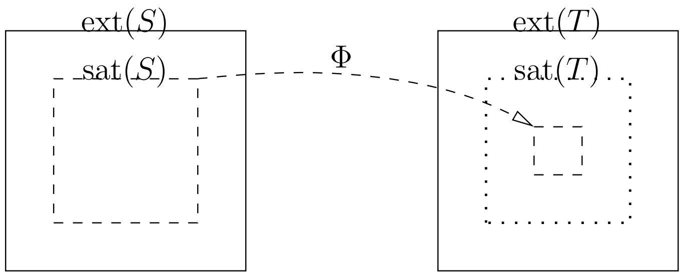
The property dependency preservation implies that $\Psi$ is a weak reduction. So every image $\Psi^*(f_T)$ of an instance $f_T \in \mathrm{sat}(T)$ of $T$ is an instance of $S$, $\Psi^*(f_T) \in \mathrm{sat}(S)$. Figure 6.1(c) shows this situation. The rectangle drawn with dotted lines on the right side of the diagram represents $\mathrm{sat}(T)$. This rectangle is mapped by $\Psi$ or better its induced mapping $\Psi^*$ onto the rectangle with dotted boundaries on the left side. It lies completely in the dashed rectangle. The latter rectangle depicts $\mathrm{sat}(S)$.

In Figure 6.1(d) we draw the situation for translation refinements. We obtain this diagram by thinking of Figures 6.1(a) and 6.1(b) as transparent slides and laying these slides one over the other.[2] Again the translation scheme $\Phi$ has translation scheme $\Psi$ as left inverse.

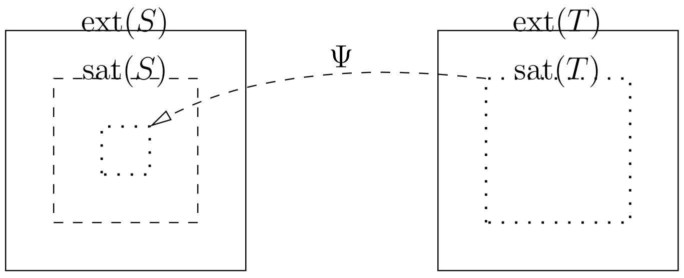
[2]The order is irrelevant.



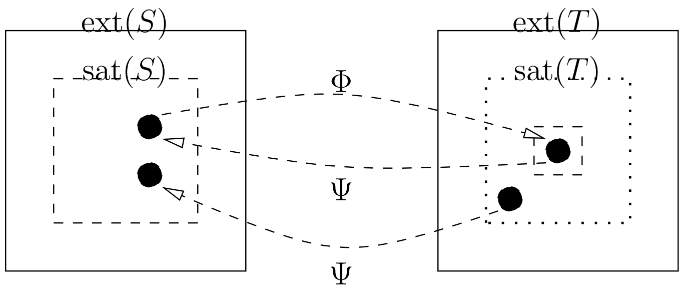
(a) Information preserving translation scheme (Def. 6.16)



(b) Weak reduction (Def. 6.14)



(c) Dependency preserving translation scheme (Def. 6.18)



(d) Translation refinement (Def. 6.19)

Figure 6.1: Translation schemes

For every instance $f_S$ of the schema $S$, the translation scheme $\Psi$ is the left inverse such that $\text{compl}_{\Psi^*(\Phi^*(D))}(\Psi^*(\Phi^*(f_S))) = \text{compl}_S(f_S)$. Additionally, we know that every instance $f_T \in \text{sat}(T)$, be it an image of any instance $f_S \in \text{sat}(S)$ or not, is mapped onto an instance of $\text{sat}(S)$ by mapping $\Psi^*$. This knowledge is acquired from the property that $\Psi$ is a weak reduction from $\text{sat}(T)$ to $\text{sat}(S)$. The phrase "or not" in the sentence preceding the last sentence is captured by the lower most arrow labelled $\Psi$.

Next we discuss Qian's approach. We can graphically represent the property for information-preserving transformations as follows. If a translation scheme is a constraint-preserving transformation every instance $f_S \in \text{sat}(S)$ is mapped onto an instance of $T$, $\Phi^*(f_S) \in \text{sat}(T)$, (cf. Fig. 6.2(a)). The big dashed rectangle on the left side is mapped onto the small one on the right side. The small rectangle lies in the rectangle with the dotted boundaries, which represents $\text{sat}(T)$, because the set of images of $\text{sat}(S)$ is a subset of $\text{sat}(T)$.

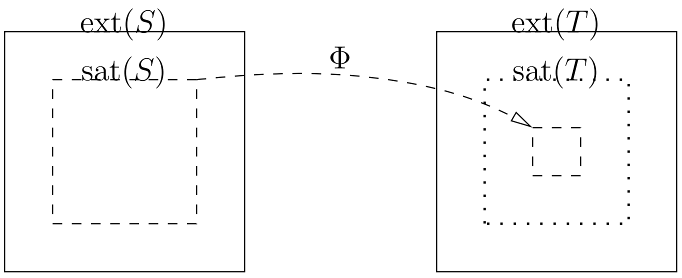
Figure 6.2(b) captures the situation when $\Phi$ is an instance-preserving transformation. Every instance $f_T \in \text{sat}(T)$ is an image of an instance $f_S \in \text{sat}(S)$ of $S$, $f_T = \Phi^*(f_S)$. This means graphically that the image of the smaller dashed rectangle on the left side spans the big one on the right side, which represents $\text{sat}(T)$.

Having an information-preserving transformation, we can describe the situation as in Fig. 6.2(c). We inflate the small dashed rectangles in Figures 6.2(a) and 6.2(b) to the size of the respective surrounding dotted rectangles. Technically, $\Phi^*$ is a total and onto mapping from $\text{sat}(S)$ to $\text{sat}(T)$.

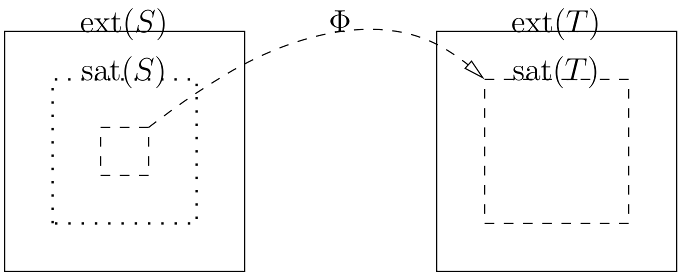
Finally, we capture the properties for view instance support in Fig. 6.3(a). A schema $S$ provides view instance support if the set of images of instances $f_S \in \text{sat}(S)$ of $S$ includes the set of instances $\text{sat}(T)$ of $T$. Therefore the dashed rectangle representing the images of instances of $\text{sat}(S)$ surrounds the set of instances $\text{sat}(T)$. The arrow is labelled with $Q^{\Phi}$. This notation indicates that the induced mapping $\Phi^*$ of a translation scheme $\Phi$ can be regarded as query over schema $S$.

At this stage we commence a comparison between the three approaches, although we have not completely described view instance support. The comparison is appropriate at this point, because matters are simple. Comparing Fig. 6.2(b) and Fig. 6.3(a), we discover that the description can be restated such that it is equivalent to the property of being an instance-preserving transformation. When we deflate the right dashed rectangle to the size of the dotted rectangle in Fig. 6.3(a), the deflation is accompanied by deflating the dashed rectangle on the left-hand side. The side-effect is that a dotted rectangle appears again to represent $\text{sat}(S)$. The equality of both notations signifies that if $\Phi$ is an instance-preserving transformation, then $S$ supports $T$ with supporting query $Q^{\Phi}$. The converse does not hold since not every query can be expressed as a translation scheme, provided the query language is sufficiently powerful.

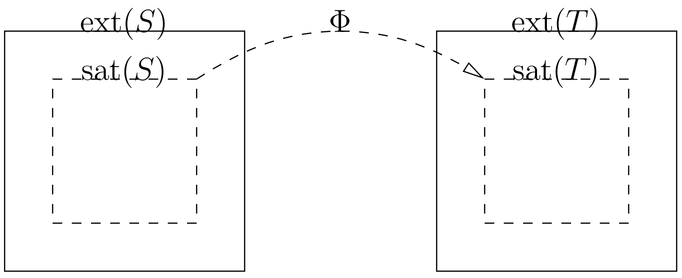
The case that $S$ supports $T$ faithfully is depicted in Fig. 6.3(b). In graphical terms that means that the dotted rectangle for $\text{sat}(T)$ is increased up to its boundaries, which are set by the rectangle drawn with dashed lines. Comparing this time Figures 6.2(c) and 6.3(b), we see they are identical. This identity means that if $\Phi$ is an information-preserving transformation from $S$ to $T$, $S$ provides faithful view instance support for $T$.



(a) Constraint-preserving transformation (Def. 6.22)



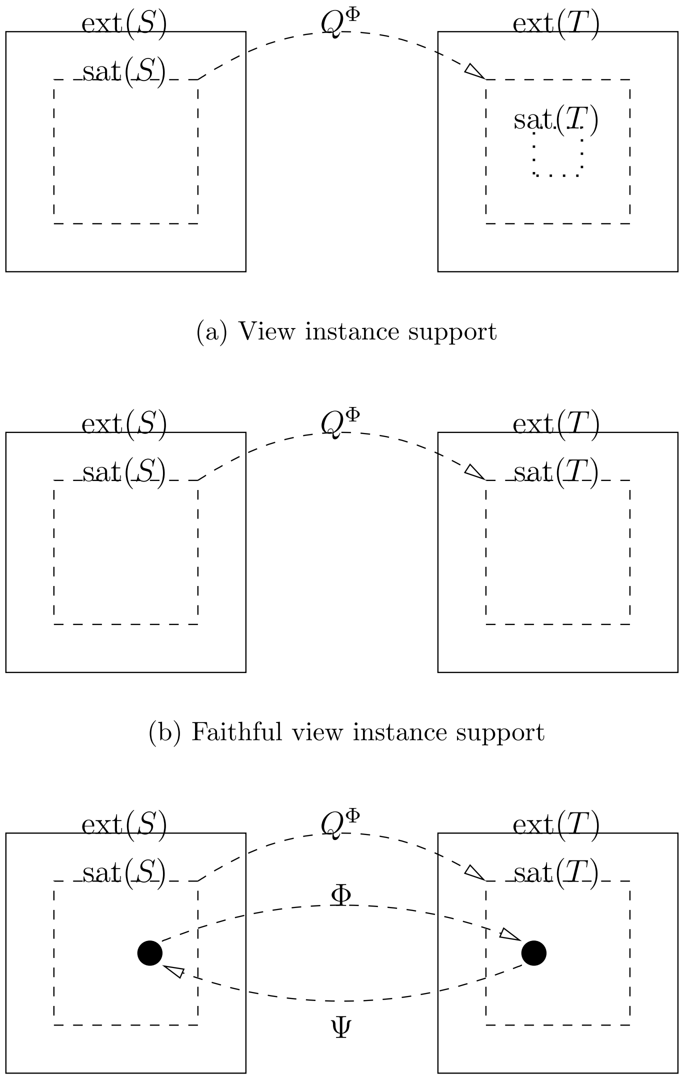
(b) Instance-preserving transformation (Def. 6.23)



(c) Information-preserving transformation (Def. 6.24)

Figure 6.2: Signature Interpretations

When $S$ provides even unique view instance support the supporting query is injective. Again thinking of the figures as slides, we can employ our overlay technique using Figures 6.1(a) and 6.3(b), and end up with Fig. 6.3(c). This case can be expressed neither in our framework nor in the notation of Qian. Figure 6.1(d) reveals that the notion translation refinement comes close to unique view instance support. If we require that every instance of $T$ is an image of some instance of $S$, Figures 6.1(d) and 6.3(c) are identical. We reach exactly this goal when we revert to the notation of Qian. When a translation scheme $\Phi$ is additionally an instance-preserving transformation, the schema $S$ provides unique view instance support for schema $T$.



(a) View instance support

(b) Faithful view instance support

(c) Unique view instance support

Figure 6.3: View instance support (Def. 6.26)

When we collate our notation and Qian's notation, we spot the similarity of dependency preservation and constraint-preserving transformations. This similarity is in effect equality. The difference lies in the application. While dependency preservation is used in the definition of translation refinements in the direction from $T$ to $S$, a constraint-preserving transformation is used in the definition of information-preserving transformations from $S$ to $T$.

We can conceive schemas $S$ and $T$ such that there is an information-preserving transformation from $S$ to $T$, but we never find a translation scheme to show that $T$ is a translation refinement of $S$. The idea is that we need schemas $S$ and $T$ whose transition from $S$ to $T$ involves an unrecoverable loss of information. So let $S$ be a schema with $\mathrm{HIER}_S := \{c{::}d\}$ and $\Phi$ be a translation scheme with $\phi_{\mathrm{HIER}}(C, D) := C \overset{\circ}{=} D$ and $\phi_{\mathrm{pop}}(O, C) := O{:}C \wedge C \overset{\circ}{=} d$. So the class hierarchy is flat. Formula $\phi_{\mathrm{HIER}}(C, D)$ is deliberately chosen in a way such that $C$ and $D$ are free variables but the formula is only true for assignments $\nu$ such that $\nu(C) = \nu(D)$. Formula $\phi_{\mathrm{pop}}(O, C)$ populates only the class $d$ with the objects that were an element of class $d$ in the original extension. The translation scheme $\Phi$ is a translation scheme from $S$ to a schema $T$ such that $\mathrm{HIER}_T = \emptyset$. The translation scheme suppresses the information that some objects of class $d$ are also objects of class $c$. We assume further that $S$ and $T$ do not contain application-dependent semantic constraints. We can see that the translation scheme $\Phi$ is a constraint-preserving and instance-preserving transformation from $S$ to $T$, but the translation scheme $\Phi$ is not a witness for the fact that the schema $T$ is a refinement of schema $S$. We can even find no way to define $\Phi$, more specific $\phi_{\mathrm{pop}}(O, C)$ such that there is a left inverse on $\mathrm{sat}(S)$. We can construe every instance of $T$ as instance of $S$. But still the set of instances of $S$ is larger than the set of instances of $T$, because there are instances that are identical to instances of $T$ with respect to the population of class $d$. They differ in the population of class $c$.

We conclude this section by showing that we can track down two schemas $S$ and $T$ such that the schema $T$ is a translation refinement of the schema $S$, but there is no information-preserving transformation from the schema $S$ to the schema $T$. This time we choose a schema $T$ whose information capacity is higher than the information capacity of the schema $S$. So let $S$ be a schema with $\mathrm{SIG}_S := \{c[x \Rightarrow c]\}$ and $T$ be a schema with $\mathrm{SIG}_T := \{c[x \Rightarrow c; y \Rightarrow c]\}$. For these schemas the translation scheme $\Phi$ with $\phi_{\Rightarrow}(M, C, R) := (M \overset{\circ}{=} x \vee M \overset{\circ}{=} y) \wedge C \overset{\circ}{=} c \wedge R \overset{\circ}{=} c$ and $\phi_{\rightarrow}(M, O, R) := (M \overset{\circ}{=} x \vee M \overset{\circ}{=} y) \wedge O{:}c[x \rightarrow R]$ and with the left inverse $\Psi$ with $\psi_{\Rightarrow}(M, C, R) := M \overset{\circ}{=} x \wedge C \overset{\circ}{=} c \wedge R \overset{\circ}{=} c$ and with $\psi_{\rightarrow}(M, O, R) := M \overset{\circ}{=} x \wedge O{:}c[x \rightarrow R]$ are witness that the schema $T$ is a translation refinement of the schema $S$. In this case it is impossible to stumble upon a translation scheme that is instance-preserving. The number of instances of the schema $T$ is the number of instances of the schema $S$ to the power of 2.

# Chapter 7

# Pivoting: A Database Transformation

Having defined our data model and knowing how to obtain a schema in this data model, we want this schema to possess some nice properties. We derive these properties from properties required of data management systems. We mention the latter properties in the introduction. A database schema should not structure redundant data, and should support cheap insertion, deletion and update operations on its instances. In this chapter we propose a database transformation that has the potential to improve database schemas along with their instances. An improvement means that the resulting schema exhibits more "nice properties" than the original schema. We call this database transformation *attribute pivoting.* A variant of it has been introduced in [BMPS96] under the name *property pivoting.* We altered the name to attribute pivoting, because what is called a property in [BMPS96] is called an attribute in this work. Attribute pivoting as well as property pivoting are special cases of the transformation *pivoting* presented in [BMP96]. Since we do not refer to (general) pivoting in this work, we use the term pivoting meaning attribute pivoting.

Our entry point to achieving nice properties are formalisations of relationships. The features of object-oriented data models offer a wide range of possibilities to formalise relationships between entities of the real world. The algorithm in Chap. 5 formalises entity sets simply as (entity) classes the basic types of which are determined by the pertinent properties of the entities. In order to formalise a relationship set, the relational approach is simulated. For every individual relationship an object that is counterpart to the corresponding tuple in the relational approach is constructed. Then these objects are understood as instances of a (relationship) class.

One goal is to identify redundancy in instances of (relationship) classes at design time. We use application-dependent semantic constraints for the identification of potential redundancy. Our objective is to eliminate the potential redundancy by decomposing a (relationship) class into smaller fragments. Roughly speaking, we transplant attributes from one class to another class. The classes chosen to receive new attributes are called *pivot classes.*

Pivoting is guided by application-dependent semantic constraints. The effect of the transformation is not automatically redundancy removing. We resume this issue later again.

A second effect of pivoting is its impact on semantic constraints. We see in Sections 7.4 and 7.5 that pivoting is capable to transform certain schemas with onto constraints and path functional dependencies into schemas with only onto constraints.

In the remainder of this chapter, we assume that the set $\mathrm{SC}_D$ of semantic constraints consists only of proper onto constraints and path functional dependencies.

## 7.1 Pivoting

Now we define pivoting in detail, i. e., we define how we can obtain the target schema and the view instance supporting query for a given source schema. The effect of pivoting is concentrated on two classes and their attributes. Above we talked about transplanting attributes to the pivot class. This is not quite correct. Instead we introduce a new subclass of the pivot class and perform all operations on this subclass. The introduction of the new subclass enables a smoother treatment of pivoting in connection with onto constraints, and we often use the term pivot class for both the original pivot class and the newly introduced subclass.

Definition 7.1 (Pivoting)
*Let $S = \langle \mathrm{CLASS}_S | \mathrm{METH}_S | \mathrm{HIER}_S | \mathrm{SIG}_S | \mathrm{SC}_S \rangle$ be a schema, $p$ be a constant symbol not used elsewhere, $c \in \mathrm{CLASS}_S$ be a class, $\mathbb{M} \subset \mathrm{Attr}_S(c)$ be a non-empty set of proper attributes for $c$, and $m_p \in (\mathrm{Attr}_S(c) \backslash \mathbb{M})$ be a proper attribute for $c$, called* pivot attribute, *with the unique result class $c_p$, called* pivot class.
*We assume that $c\{m_p | c_p\} \notin \mathrm{OC}_S^{+s}$ otherwise we will not have to introduce the class $p$. Then the pivoted schema $T := \langle \mathrm{CLASS}_T | \mathrm{METH}_T | \mathrm{HIER}_T | \mathrm{SIG}_T | \mathrm{SC}_T \rangle$ is obtained as follows:*

- *For introducing a new subclass, we add the constant symbol $p$ to the set of classes,*

  $$\mathrm{CLASS}_T := \mathrm{CLASS}_S \cup \{p\} \ ,$$

  *and we make the new class $p$ a subclass of the pivot class $c_p$,*

  $$\mathrm{HIER}_T := \mathrm{HIER}_S \cup \{p :: c_p\} \ .$$

- *For transplanting attributes, we introduce new attributes and remove the obsolescent ones. The constant symbols for denoting these new attributes are chosen in a way that it is possible to keep track of the transformation process and avoid name clashes with already existing symbols,*

  $$\mathrm{METH}_T := (\mathrm{METH}_S \backslash \mathbb{M}) \cup \{c\_m_p\_m \mid m \in \mathbb{M}\} \ ,$$

  *and we add appropriate signatures $p[c\_m_p\_m \Rightarrow c_m]$ for all attributes $m \in \mathbb{M}$ and for their result classes $c_m$,*

  $$\{p[c\_m_p\_m \Rightarrow c_m] \mid m \in \mathbb{M} \textit{ and } \mathrm{SIG}_S \models c[m \Rightarrow c_m]\} \ ,$$

*and, accordingly, we remove the obsolescent signatures for class $c$ of the form $c[m \Rightarrow c_m]$ for all attributes $m \in \mathbb{M} \cup \{m_p\}$ and their result classes $c_m$,*

$$\mathrm{SIG}_S \backslash \{c[m \Rightarrow c_m] \mid m \in \mathbb{M} \cup \{m_p\} \textit{ and } c_m \in \mathrm{CLASS}_S\} \ .$$

*Finally we add the signature $c[m_p \Rightarrow p]$ in order to enable navigation to the new subclass $p$, and further onto the pivoted attributes, from the class $c$,*

$$\{c[m_p \Rightarrow p]\} \ .$$

*Thus the new signatures are summarised as*

$$\begin{aligned} \mathrm{SIG}_T := & \{p[c\_m_p\_m \Rightarrow c_m] \mid m \in \mathbb{M} \textit{ and } \mathrm{SIG}_S \models c[m \Rightarrow c_m]\} \cup \\ & \mathrm{SIG}_S \backslash \{c[m \Rightarrow c_m] \mid m \in \mathbb{M} \cup \{m_p\} \textit{ and } c_m \in \mathrm{CLASS}_S\} \cup \\ & \{c[m_p \Rightarrow p]\} \ . \end{aligned}$$

- *For adjusting the semantic constraints, we replace the obsolescent attributes $m \in \mathbb{M}$ by the new navigational paths $m_p.c\_m_p\_m$ wherever they occur in the set of path functional dependencies,*

$$\mathrm{PFD}_T := \mathrm{PFD}_S[m_p.c\_m_p\_m/m \mid m \in \mathbb{M}] \ .$$

*Correspondingly, for an onto constraint of the form $c\{m|c_m\}$ with an obsolescent attribute $m$, we replace $c$ by the new subclass $p$ and $m$ by the new attribute $c\_m_p\_m$. We add the onto constraint $c\{m_p|p\}$ as a means to ensure the equivalence of both schemas,*

$$\begin{aligned} \mathrm{OC}_T := & \{p\{c\_m_p\_m|c_m\} \mid m \in \mathbb{M} \textit{ and } c\{m|c_m\} \in \mathrm{OC}_S\} \cup \\ & \{d\{m|c_m\} \in \mathrm{OC}_S \mid m \notin \mathbb{M}\} \cup \\ & \{c\{m_p|p\}\} \ . \end{aligned}$$

*For treating the original schema $S$ as a view on the pivoted schema $T$, we define a query $Q_{T \to S}$ that collapses a new navigational path $m_p.c\_m_p\_m$ to the obsolescent attribute $m$, for $m \in \mathbb{M}$, and suppresses objects of the new subclass $p$, while leaving the rest of an instance unchanged,*

$$\begin{aligned} Q_{T \to S} \ := \ & \{O{:}d \longleftarrow O{:}d \mid d \in \mathrm{CLASS}_S\} \cup & (7.1) \\ & \{O[m \to R] \longleftarrow O[m \to R] \mid m \in (\mathrm{METH}_S \backslash \mathbb{M})\} \cup & (7.2) \\ & \{O[m \to R] \longleftarrow O{:}c[m_p \to P] \wedge P{:}p[c\_m_p\_m \to R] \mid m \in \mathbb{M}\} \ . & (7.3) \end{aligned}$$

*For treating the pivoted schema $T$ as a view on the original schema $S$, we define a query $Q_{S \to T}$ that retains the class population and the values for most attributes. The obsolescent attributes and their values are transplanted to the pivot class $p$ and its objects, respectively. This class is populated with objects of class $c_p$ that are referenced by an object of class $c$ via the pivot attribute $m_p$.*

$$\begin{aligned} Q_{S \to T} \ := \ & \{O{:}d \longleftarrow O{:}d \mid d \in \mathrm{CLASS}_S\} \cup & (7.4) \\ & \{O[m \to R] \longleftarrow O[m \to R] \mid m \in (\mathrm{METH}_S \backslash \mathbb{M})\} \cup & (7.5) \\ & \{P{:}p[c\_m_p\_m \to R] \longleftarrow O{:}c[m_p \to P; m \to R] \mid m \in \mathbb{M}\} & (7.6) \end{aligned}$$

*Finally, we define the tuple* $\mathrm{piv}(S, p, c, \mathbb{M}, m_p) := (T, Q_{T \to S}, Q_{S \to T})$.

To give an impression of how pivoting works, we perform pivoting on the examples that have been accompanying us throughout this work. First comes the "good" example.

Example 7.2
*We perform pivoting on the schema in Fig. 5.1. To get a smoother exposition, we dispense with the introduction of cryptic method names. We stick to the original method names instead. This is possible, because no name clashes can occur in this example. Additionally, we include four onto constraints.*

$$\begin{aligned}&\mathsf{Assignment}\{\mathsf{teacher}|\mathsf{Teacher}\}\\&\mathsf{Assignment}\{\mathsf{course}|\mathsf{Course}\}\\&\mathsf{Assignment}\{\mathsf{room}|\mathsf{Room}\}\\&\mathsf{Assignment}\{\mathsf{wing}|\mathsf{Wing}\}\end{aligned}$$

*These onto constraints are formalisations of the restrictions that*

- *every* Teacher *has to give a* Course*,*
- *if there is a* Course*, we have some information about it,*
- *every* Room *is occupied,*
- *we have a* Teacher *in every* Wing*.*

*We perform pivoting with the pivot attribute* course *and the pivoted attributes* assistant *and* date*. Since we assume the presence of the onto constraint*

$$\mathsf{Assignment}\{\mathsf{course}|\mathsf{Course}\}\ ,$$

*the introduction of the subclass becomes unnecessary. (Example 7.3 shows how that is done.) Therefore we keep the set of classes and the class hierarchy unchanged.*

$$\begin{aligned}\mathrm{CLASS} &:= \{\mathsf{String}, \mathsf{Int}, \mathsf{Assignment}, \mathsf{Course}, \mathsf{Teacher}, \mathsf{Assistant}, \mathsf{Date}, \mathsf{Room}, \mathsf{Wing}\}\\ \mathrm{HIER} &:= \emptyset\end{aligned}$$

*Next we consider the set of attributes. We refrain from the introduction of new cryptic method names, therefore even the set of methods remains unaltered.*

$$\begin{aligned}\mathrm{METH} := \{&\mathsf{course}, \mathsf{teacher}, \mathsf{assistant}, \mathsf{room}, \mathsf{wing}, \mathsf{title}, \mathsf{te\_name}, \mathsf{as\_name}, \mathsf{ro\_name},\\ &\mathsf{date}, \mathsf{year}, \mathsf{month}, \mathsf{day}, \mathsf{size}, \mathsf{address}\}\end{aligned}$$

*When we look at the set of signatures, we have to declare the signatures for the attributes* assistant *and* date *on the pivot class* Course*.*

$$\begin{aligned}\mathsf{Course}[&\mathsf{assistant} \Rightarrow \mathsf{Assistant};\\ &\mathsf{date} \Rightarrow \mathsf{Date}]\end{aligned}$$

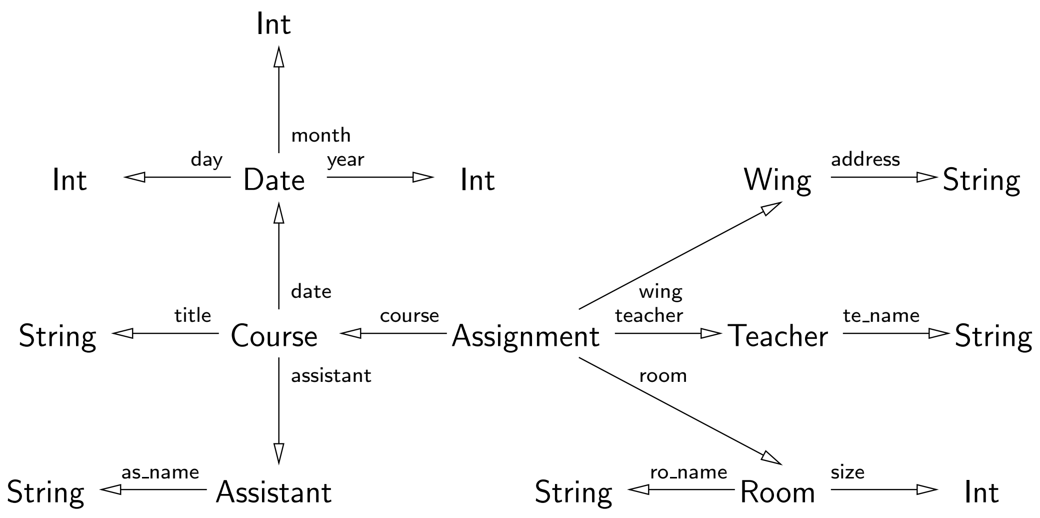


Figure 7.1: First applying pivoting with course as pivoted attribute

*The old declarations for these attributes are obliterated. Finally, the redirection of result classes of the pivot attribute* course *is unnecessary, because we do not introduce a subclass of the pivot class. The resulting schema graph is shown in Fig. 7.1.*

$$
\begin{aligned}
\mathrm{SIG} := \{&\mathsf{Course}[\mathsf{title} \Rightarrow \mathsf{String};\\
&\qquad\quad \mathsf{assistant} \Rightarrow \mathsf{Assistant};\\
&\qquad\quad \mathsf{date} \Rightarrow \mathsf{Date}],\\
&\mathsf{Teacher}[\mathsf{te\_name} \Rightarrow \mathsf{String}],\\
&\mathsf{Assistant}[\mathsf{as\_name} \Rightarrow \mathsf{String}],\\
&\mathsf{Date}[\mathsf{year} \Rightarrow \mathsf{Int}; \mathsf{month} \Rightarrow \mathsf{Int}; \mathsf{day} \Rightarrow \mathsf{Int}],\\
&\mathsf{Room}[\mathsf{size} \Rightarrow \mathsf{Int}; \mathsf{ro\_name} \Rightarrow \mathsf{String}],\\
&\mathsf{Wing}[\mathsf{address} \Rightarrow \mathsf{String}],\\
&\mathsf{Assignment}[\mathsf{course} \Rightarrow \mathsf{Course};\\
&\qquad\qquad\quad \mathsf{teacher} \Rightarrow \mathsf{Teacher};\\
&\qquad\qquad\quad \mathsf{room} \Rightarrow \mathsf{Room};\\
&\qquad\qquad\quad \mathsf{wing} \Rightarrow \mathsf{Wing}]\}
\end{aligned}
$$

*For adjusting the semantic constraints, we replace the attributes* assistant *and* date *occurring in path functional dependencies by the "path functions"* course.assistant *and* course.date*, respectively. The onto constraints remain unchanged.*

$$
\begin{aligned}
\mathrm{SC} := \mathrm{AX} \cup \\
&\{\mathsf{Assignment}(.\mathsf{course} \rightarrow .\mathsf{course.assistant}\quad .\mathsf{course.date}),\\
&\ \mathsf{Assignment}(.\mathsf{teacher} \rightarrow .\mathsf{room}),\\
&\ \mathsf{Assignment}(.\mathsf{room} \rightarrow .\mathsf{wing}),\\
&\ \mathsf{Assignment}(.\mathsf{course}\quad .\mathsf{teacher} \rightarrow .\mathrm{Id}),\\
&\ \mathsf{Assignment}\{\mathsf{teacher}|\mathsf{Teacher}\},\\
&\ \mathsf{Assignment}\{\mathsf{course}|\mathsf{Course}\},\\
&\ \mathsf{Assignment}\{\mathsf{room}|\mathsf{Room}\},\\
&\ \mathsf{Assignment}\{\mathsf{wing}|\mathsf{Wing}\}\}
\end{aligned}
$$

*The view instance supporting query $Q_{T \to S}$ is*

$$
\begin{aligned}
Q_{T\to S} := \{&O{:}\mathsf{String} \longleftarrow O{:}\mathsf{String}, O{:}\mathsf{Int} \longleftarrow O{:}\mathsf{Int},\\
&O{:}\mathsf{Assignment} \longleftarrow O{:}\mathsf{Assignment}, O{:}\mathsf{Course} \longleftarrow O{:}\mathsf{Course},\\
&O{:}\mathsf{Teacher} \longleftarrow O{:}\mathsf{Teacher}, O{:}\mathsf{Assistant} \longleftarrow O{:}\mathsf{Assistant},\\
&O{:}\mathsf{Date} \longleftarrow O{:}\mathsf{Date}, O{:}\mathsf{Room} \longleftarrow O{:}\mathsf{Room}, O{:}\mathsf{Wing} \longleftarrow O{:}\mathsf{Wing}\}\\
\cup&\\
\{&O[\mathsf{course} \to R] \longleftarrow O[\mathsf{course} \to R], O[\mathsf{teacher} \to R] \longleftarrow O[\mathsf{teacher} \to R]\\
&O[\mathsf{room} \to R] \longleftarrow O[\mathsf{room} \to R], O[\mathsf{wing} \to R] \longleftarrow O[\mathsf{wing} \to R],\\
&O[\mathsf{title} \to R] \longleftarrow O[\mathsf{title} \to R], O[\mathsf{te\_name} \to R] \longleftarrow O[\mathsf{te\_name} \to R],\\
&O[\mathsf{as\_name} \to R] \longleftarrow O[\mathsf{as\_name} \to R],\\
&O[\mathsf{ro\_name} \to R] \longleftarrow O[\mathsf{ro\_name} \to R],\\
&O[\mathsf{year} \to R] \longleftarrow O[\mathsf{year} \to R], O[\mathsf{month} \to R] \longleftarrow O[\mathsf{month} \to R],\\
&O[\mathsf{day} \to R] \longleftarrow O[\mathsf{day} \to R], O[\mathsf{size} \to R] \longleftarrow O[\mathsf{size} \to R],\\
&O[\mathsf{address} \to R] \longleftarrow O[\mathsf{address} \to R]\}\\
\cup&\\
\{&O[\mathsf{assistant} \to R] \longleftarrow O{:}\mathsf{Assignment}[\mathsf{course} \to P] \wedge P : \mathsf{Course}[\mathsf{assistant} \to R],\\
&O[\mathsf{date} \to R] \longleftarrow O{:}\mathsf{Assignment}[\mathsf{course} \to P] \wedge P : \mathsf{Course}[\mathsf{date} \to R]\}\ ,
\end{aligned}
$$

*where $S$ is the schema* Assignment *and $T$ the pivoted schema.*

Now comes the "bad" example from the introduction.

Example 7.3
*To demonstrate pivoting, we perform it on class* Phone-admin *of the schema in Fig. 5.3 We reduce the left side of the key path functional dependency*

$$\mathsf{Phone\text{-}admin}(.\mathsf{fac}\quad .\mathsf{sch}\quad .\mathsf{ph}\quad .\mathsf{dep} \rightarrow .\mathrm{Id})$$

*to* .fac *and get the key path functional dependency*

$$\mathsf{Phone\text{-}admin}(.\mathsf{fac} \rightarrow .\mathrm{Id})\ .$$

*The reduction is possible, because the attribute* fac *determines all other attributes functionally.*

*The pivot class is the class* Faculty*. As pivoted attributes we choose the attributes* sch *and* dep*. To render the example more readable, we do not introduce the cryptic names of attributes. Instead we use the original names. As subclass for the pivot class* Faculty *we introduce the class* Phoning-faculty*. It is populated with the objects that are referenced by* Phone-admin*-objects via attribute* fac*.*

*The outcome is then the schema* Phone-Admin-Piv := $\langle \mathrm{CLASS}|\mathrm{METH}|\mathrm{HIER}|\mathrm{SIG}|\mathrm{SC}\rangle$ *with*

$$\begin{aligned}
\mathrm{CLASS} &:= \{\mathsf{Phone\text{-}admin}, \mathsf{Faculty}, \mathsf{Phoning\text{-}faculty}, \mathsf{School}, \mathsf{Phone}, \mathsf{Department}\}\\
\mathrm{METH} &:= \{\mathsf{fac}, \mathsf{sch}, \mathsf{ph}, \mathsf{dep}\}\\
\mathrm{HIER} &:= \{\mathsf{Phoning\text{-}faculty}::\mathsf{Faculty}\}\\
\mathrm{SIG} &:= \{\mathsf{Phone\text{-}admin}[\mathsf{fac} \Rightarrow \mathsf{Phoning\text{-}faculty}; \mathsf{ph} \Rightarrow \mathsf{Phone}],\\
&\qquad \mathsf{Phoning\text{-}faculty}[\mathsf{sch} \Rightarrow \mathsf{School}; \mathsf{dep} \Rightarrow \mathsf{Department}]\,\}\\
\mathrm{SC} &:= \mathrm{AX} \cup\\
&\qquad \big\{\mathsf{Phone\text{-}admin}(.\mathsf{fac} \longrightarrow .\mathsf{fac.sch}),\\
&\qquad \mathsf{Phone\text{-}admin}(.\mathsf{fac} \longrightarrow .\mathsf{ph}),\\
&\qquad \mathsf{Phone\text{-}admin}(.\mathsf{fac.sch} \longrightarrow .\mathsf{fac.dep}),\\
&\qquad \mathsf{Phone\text{-}admin}(.\mathsf{ph} \longrightarrow .\mathsf{fac.dep}),\\
&\qquad \mathsf{Phone\text{-}admin}(.\mathsf{fac} \longrightarrow .\mathrm{Id}),\\
&\qquad \mathsf{Phone\text{-}admin}\{\mathsf{fac}|\mathsf{Phoning\text{-}faculty}\}\big\}\ .
\end{aligned}$$

*The transformation pivoting removes in the class* Phone-admin *the declarations for the attributes* sch *and* dep*. These declarations occur again as declarations for the class* Phoning-faculty*. This change in the signature declarations has to be reflected by the path functional dependencies. For example, whenever the path function* .sch *appears in a path functional dependency, we replace it by the path function* .fac.sch*. This replacement is done for the path functional dependency*

$$\mathsf{Phone\text{-}admin}(.\mathsf{fac} \longrightarrow .\mathsf{sch})$$

*and leads to the path functional dependency*

$$\mathsf{Phone\text{-}admin}(.\mathsf{fac} \longrightarrow .\mathsf{fac.sch})\ .$$

*We do not have to change any onto constraints in this example, because there is no onto constraint in the original schema. But we have to introduce the onto constraint*

$$\mathsf{Phone\text{-}admin}\{\mathsf{fac}|\mathsf{Phoning\text{-}faculty}\}\ .$$

*The schema graph for the pivoted schema* Phone-Admin-Piv *can be found in Fig. 7.2.*

## 7.2 Pivoting and Equivalence

The next theorem gives us the answer to the question of when a pivoted schema captures as much information as the original schema, i. e., when it provides unique view instance support, in the sense of Def. 6.26, and vice versa.

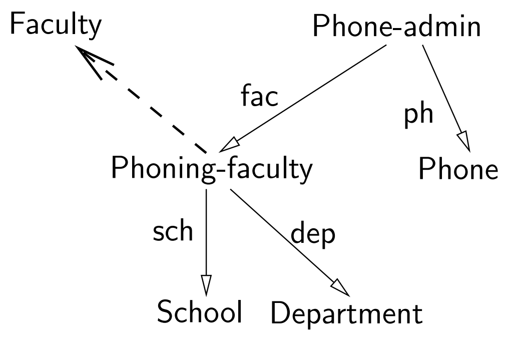


Figure 7.2: Schema graph of the pivoted schema

THEOREM 7.4
*Let $S$ be a schema, and the vector $(T, Q_{T\to S}, Q_{S\to T}) := \mathrm{piv}(S, p, c, \mathbb{M}, m_p)$ be the outcome of pivoting the schema $S$. Then the following statements are equivalent.*

1. *The schema $T$ provides unique view instance support for the schema $S$ with supporting query $Q_{T\to S}$.*

2. *The schema $S$ provides unique view instance support for the schema $T$ with supporting query $Q_{S\to T}$.*

3. *All pivoted attributes are elements of the closure of the pivot attribute, $\mathbb{M} \subset \{.m_p\}^{+_{S,c}}$.*

PROOF FOR 1 $\Longrightarrow$ 2. Since the schema $T$ provides unique view instance support for the schema $S$ with query $Q_{T\to S}$, the equation

$$\{\mathrm{compl}_S(f_S) \mid f_S \in \mathrm{sat}(S)\} = \{\mathrm{compl}_S(\mathrm{eval}_T(Q_{T\to S}, f_T)) \mid f_T \in \mathrm{sat}(T)\} \quad (7.7)$$

holds and the query $Q_{T\to S}$ is injective on the saturated elements of $\mathrm{sat}(S)$. We have to show that the equation

$$\{\mathrm{compl}_T(\mathrm{eval}_S(Q_{S\to T}, f_S)) \mid f_S \in \mathrm{sat}(S)\} = \{\mathrm{compl}_T(f_T) \mid f_T \in \mathrm{sat}(T)\} \quad (7.8)$$

holds and the query $Q_{S\to T}$ is injective on the saturated elements of $\mathrm{sat}(T)$.

We show first that the query $Q_{S\to T}$ is the inverse of the query $Q_{T\to S}$ on the saturated elements of $\mathrm{sat}(T)$.

To show that the query $Q_{S\to T}$ is the inverse of the query $Q_{T\to S}$ on the saturated elements of $\mathrm{sat}(T)$, we have to prove the following equation

$$\mathrm{compl}_T(f_T) = \mathrm{compl}_T(\mathrm{eval}_S(Q_{S\to T}, \mathrm{eval}_T(Q_{T\to S}, f_T))) \ ,$$

and that the chaining of the query evaluations is well-defined.

The rules in (7.1), (7.2) and (7.4), (7.5) are equal and constitute the projection query on the schema $T$ and $S$, respectively. By Lem. 3.17, these rules ensure for all classes $d \in \mathrm{CLASS}_S$

$$o{:}d \in \mathrm{compl}_T(f_T) \text{ iff } o{:}d \in \mathrm{eval}_T(Q_{T\to S}, f_T)$$

and $o \in \mathcal{F}_0 \backslash (\mathrm{CLASS}_S \cup \mathrm{METH}_S)$, and for all attributes $m \in \mathrm{METH}_S \backslash \mathbb{M}$

$$o[m \to r] \in \mathrm{compl}_T(f_T) \text{ iff } o[m \to r] \in \mathrm{eval}_T(Q_{T\to S}, f_T)$$

and $o, r \in \mathcal{F}_0 \backslash (\mathrm{CLASS}_S \cup \mathrm{METH}_S)$, i. e. the restrictions on molecular formulae in an extension of the schema $S$ are met.

If $o_p{:}p[c\_m_p\_m \to r] \in \mathrm{compl}_T(f_T)$ for an attribute $m \in \mathbb{M}$, then, due to the fact that $f_T$ is an instance of the schema $T$ satisfying the onto constraint $c\{m_p|p\}$, there exists an object $o_c$ element of the class $c$ referencing the object $o_p$ via the attribute $m_p$ in the completion of the instance $f_T$, $o_c{:}c[m_p \to o_p] \in \mathrm{compl}_T(f_T)$. Then the instantiation of the body of a rule in (7.3),

$$o_c{:}c[m_p \to o_p] \wedge o_p{:}p[c\_m_p\_m \to r] \ ,$$

is satisfied by the completion $\mathrm{compl}_T(f_T)$, and thus $o_c[m \to r] \in \mathrm{eval}_T(Q_{T\to S}, f_T)$ and $o_c, r \in \mathcal{F} \backslash (\mathrm{CLASS}_S \cup \mathrm{METH}_S)$ hold. The fact that $o_c{:}c[m_p \to o_p] \in \mathrm{compl}_T(f_T)$ entails that $o_c{:}c[m_p \to o_p] \in \mathrm{eval}_T(Q_{T\to S}, f_T)$ since $c \in \mathrm{CLASS}_S$ and $m_p \in \mathrm{METH}_S \backslash \mathbb{M}$. Hence the instantiation of the body of a rule in (7.6) $o_c{:}c[m_p \to o_p; m \to r]$ is satisfied by $\mathrm{eval}_T(Q_{T\to S}, f_T)$ and therefore $o_p{:}p[c\_m_p\_m \to r] \in \mathrm{eval}_S(Q_{S\to T}, \mathrm{eval}_T(Q_{T\to S}, f_T))$ holds.

If $o_p{:}p[c\_m_p\_m \to r] \in \mathrm{eval}_S(Q_{S\to T}, \mathrm{eval}_T(Q_{T\to S}, f_T))$, then an instantiation of a rule in (7.6) must exist with $o_c{:}c[m_p \to o_p; m \to r]$ satisfied by $\mathrm{eval}_T(Q_{T\to S}, f_T)$. This satisfaction entails that an instantiation of a rule in (7.3) must exist with $o_p{:}p[c\_m_p\_m \to r] \in \mathrm{compl}_T(f_T)$.

Having established proof that the query $Q_{S\to T}$ is the inverse of the query $Q_{T\to S}$ on the saturated elements of $\mathrm{sat}(T)$, we prove for any two instances $f_S \in \mathrm{sat}(S)$ and $f_T \in \mathrm{sat}(T)$, if $\mathrm{compl}_S(f_S) = \mathrm{compl}_S(\mathrm{eval}_T(Q_{T\to S}, f_T))$ holds, then $\mathrm{compl}_T(\mathrm{eval}_S(Q_{S\to T}, f_S)) = \mathrm{compl}_T(f_T)$ holds as well.

Now if we assume $\mathrm{compl}_S(f_S) = \mathrm{compl}_S(\mathrm{eval}_T(Q_{T\to S}, f_T))$, then, by Lem. 3.18 this equality implies the equality $\mathrm{eval}_S(Q_{S\to T}, f_S) = \mathrm{eval}_S(Q_{S\to T}, \mathrm{eval}_T(Q_{T\to S}, f_T))$. Since the query $Q_{S\to T}$ is the inverse of the query $Q_{T\to S}$ on the saturated elements, the equality

$$\mathrm{compl}_T(\mathrm{eval}_S(Q_{S\to T}, \mathrm{eval}_T(Q_{T\to S}, f_T))) = \mathrm{compl}_T(f_T)$$

holds, and hence the equality

$$\begin{aligned}\mathrm{compl}_T(\mathrm{eval}_S(Q_{S\to T}, f_S)) &= \mathrm{compl}_T(\mathrm{eval}_S(Q_{S\to T}, \mathrm{eval}_T(Q_{T\to S}, f_T))) \\ &= \mathrm{compl}_T(f_T) \ .\end{aligned} \tag{7.9}$$

Finally, we are ready to proof (7.8). For each instance $f_T \in \mathrm{sat}(T)$ of the schema $T$, there exists due to (7.7) and (7.9) an instance $f_S \in \mathrm{sat}(S)$ of the schema $S$ such that

$$\mathrm{compl}_T(\mathrm{eval}_S(Q_{S\to T}, f_S)) = \mathrm{compl}_T(f_T) \ .$$

Likewise for each instance $f_S \in \mathrm{sat}(S)$ of the schema $S$, there exists due to (7.7) and (7.9) an instance $f_T \in \mathrm{sat}(T)$ of the schema $T$ such that

$$\mathrm{compl}_T(\mathrm{eval}_S(Q_{S\to T}, f_S)) = \mathrm{compl}_T(f_T) \ .$$

What remains is to show that the query $Q_{S\to T}$ is injective on the saturated elements of $\mathrm{sat}(T)$. This can be done by showing that the query $Q_{T\to S}$ is the inverse of the query $Q_{S\to T}$ on the saturated elements of $\mathrm{sat}(S)$, which follows the same line of reasoning as showing the query $Q_{S\to T}$ is the inverse of the query $Q_{T\to S}$ on the saturated elements of $\mathrm{sat}(T)$ as done above.

**Proof for 2 $\Longrightarrow$ 3.** Since the schema $S$ provides unique view instance support for the schema $T$ with supporting query $Q_{S\to T}$, (7.8) holds. This means $\mathrm{compl}_T(\mathrm{eval}_S(Q_{S\to T}, f_S))$ satisfies the unique name axioms of the schema $T$. Therefore for any $o_p{:}p \in \mathrm{eval}_S(Q_{S\to T}, f_S)$ only one $r$ exists with $o_p[c\_m_p\_m \to r] \in \mathrm{eval}_S(Q_{S\to T}, f_S)$, because $o_p{:}p$ and $o_p[c\_m_p\_m \to r]$ can only be results of (7.6). This means the path functional dependency $c(.m_p \to .m)$ is satisfied by the instance $f_S$.

Since all instances in $\mathrm{sat}(S)$ satisfy the path functional dependency $c(.m_p \to \mathbb{M})$ and the fact that $\mathrm{compl}_D(f)$ is a minimal model, $\mathbb{M} \subset \{.m_p\}^{+_{S,c}}$ holds by definition.

**Proof for 3 $\Longrightarrow$ 1.** We have to show that the equation

$$\{\mathrm{compl}_S(f_S) \mid f_S \in \mathrm{sat}(S)\} = \{\mathrm{compl}_S(\mathrm{eval}_T(Q_{T\to S}, f_T)) \mid f_T \in \mathrm{sat}(T)\} \quad (7.10)$$

holds and the query $Q_{T\to S}$ is injective on the saturated instances.

Let $f_S \in \mathrm{sat}(S)$ be an instance of the schema $S$. We construct for the instance $f_S$ an instance $f_T$ of the schema $T$ such that the equality $\mathrm{compl}_S(f_S) = \mathrm{compl}_S(\mathrm{eval}_T(Q_{T\to S}, f_T))$ inures. We define $f_T$ as $f_T := \mathrm{eval}_S(Q_{S\to T}, f_S)$.

First of all we show that $f_T$ is an extension of the schema $T$. Due to rules (7.4) and (7.5) and Lem. 3.17, we have for all classes $d \in \mathrm{CLASS}_S$

$$o{:}d \in \mathrm{compl}_S(f_S) \text{ iff } o{:}d \in f_T \quad (7.11)$$

and $o \in \mathcal{F}_0 \backslash (\mathrm{CLASS}_S \cup \mathrm{METH}_S)$, and for all attributes $m \in \mathrm{METH}_S \backslash \mathbb{M}$

$$o[m \to r] \in \mathrm{compl}_S(f_S) \text{ iff } o[m \to r] \in f_T \quad (7.12)$$

and $o \in \mathcal{F}_0 \backslash (\mathrm{CLASS}_S \cup \mathrm{METH}_S)$.

Due to rule (7.6) and the fact that $f_S$ is an instance, we have for all attributes $m \in \mathbb{M}$

$$o_c[m \to r] \in \mathrm{compl}_S(f_S) \text{ iff } o_p{:}p[c\_m_p\_m \to r] \in f_T \quad (7.13)$$

and $o_p, r \in \mathcal{F}_0 \setminus (\mathrm{CLASS}_S \cup \mathrm{METH}_S)$. The proof goes along the same line of reasoning as showing the query $Q_{S \to T}$ is the inverse of the query $Q_{T \to S}$ as done in $1 \Longrightarrow 2$.

The next steps to show that $f_T$ is an instance of the schema $T$ is to show that $f_T$ satisfies the unique name axioms, the definedness axioms and the well-typedness axioms.

Due to the nature of the rules in (7.4), (7.5) and (7.6) and the inference rules for F-logic, the only way to introduce a violation of the unique name axioms is by the rules (7.5) and (7.6) because of scalarity. But because of (7.12) and the fact that $f_S$ is an instance the rules in (7.5) do not introduce a violation of the unique name axioms.

A rule in (7.6) could introduce a violation of the unique name axioms, if there existed

$$o_c{:}c[m_p \to o_p; m \to r], o_c'{:}c[m_p \to o_p; m \to r'] \in \mathrm{compl}_S(f_S)\ .$$

But this is impossible because $\mathbb{M} \subset \{.m_p\}^{+_{S,c}}$ holds and $f_S$ is an instance of the schema $S$.

Due to the similarity of the schemas $S$ and $T$, and (7.11), (7.12) and (7.13) a violation of the remaining axioms cannot occur.

Then we prove that all semantic constraints in the schema, i. e. all onto constraints and path functional dependencies, are satisfied by the extension $f_T$.

Since the extensions $f_S$ and $f_T$ agree on the parts that are relevant for the satisfaction of the semantic constraints present in both schemas $S$ and $T$ and since $f_S$ is an instance of the schema $S$, the extension satisfies these semantic constraints.

We consider an onto constraint $p\{c\_m_p\_m|c_m\} \in \mathrm{OC}_T$ newly added to the semantic constraints in the schema $T$ because of the onto constraint $c\{m|c_m\} \in \mathrm{OC}_S$. Let $r$ be an object of the class $c_m$, $r{:}c_m \in \mathrm{compl}_T(f_T)$. By the construction of the extension $f_T$ and Lem. 3.17, $r{:}c_m \in f_T$ holds. Then due to (7.11) $r{:}c_m \in \mathrm{compl}_S(f_S)$ holds. But then because of the onto constraint $c\{m|c_m\} \in \mathrm{OC}_S$ and the fact that $f_S$ is an instance of the schema $S$, there exists $o_c[m \to r] \in \mathrm{compl}_S(f_S)$, which entails $o_p{:}p[c\_m_p\_m \to r] \in f_T$ by (7.13). Thus the onto constraint $p\{c\_m_p\_m|c_m\}$ is satisfied by the extension $f_T$.

When considering a path functional dependency tampered with by the transformation pivoting, we focus on the path-functions in the path functional dependency. We will show that any path described by such a path-function has the same end node as the original path. Therefore we consider an edge $o[m \to r] \in \mathrm{compl}_S(f_S)$. If $m \in \mathrm{METH}_S \setminus \mathbb{M}$, this edge is not changed on the modified extension. If $m \in \mathbb{M}$, then $o[m_p \to o_p], o_p[c\_m_p\_m \to r] \in \mathrm{compl}_T(f_T)$ by (7.11), (7.12) and (7.13). These edges are exactly described by the replacement $m_p.c\_m_p\_m$ for the attribute $m$ in the original path-function. Hence all path functional dependencies in $\mathrm{PFD}_T$ are satisfied, because $f_S$ satisfies $\mathrm{PFD}_S$.

Similarly, we can show that for each instance $f_T \in \mathrm{sat}(T)$ of the schema $T$ that $\mathrm{eval}_T(Q_{T\to S}, f_T)$ is an instance of the schema $S$, and finally,

$$\mathrm{compl}_T(\mathrm{eval}_S(Q_{S\to T}, \mathrm{eval}_T(Q_{T\to S}, f_T))) = \mathrm{compl}_T(f_T) \ .$$

Thus the query $Q_{T\to S}$ is injective with the inverse $Q_{S\to T}$.

□

## 7.3 Pivoting and Translation Schemes

Up to now we favoured the approach of Biskup (view instance support) and gave the definition of pivoting in his notation. We continue this bias in this work. An in-depth discussion of pivoting under translation schemes lies beyond the scope of this work. Nevertheless, we make a short detour and define pivoting by means of translation schemes.

Firstly, we give the translation scheme $\Phi$ from the original schema to the pivoted schema. In the definition the terms $\mathrm{class}(C)$ and $\mathrm{meth}(M)$ are abbreviations for the expressions $\left(\bigvee_{c\in\mathrm{CLASS}_S} C \overset{\circ}{=} c\right)$ and $\left(\bigvee_{m\in\mathrm{METH}_S} M \overset{\circ}{=} m\right)$, respectively. The set of pivoted attributes $\mathbb{M}$ is $\{m_1, \ldots, m_n\}$. Table 7.1 shows how we define pivoting by means of translation schemes. At first glance there seems to be no similarity between the view instance supporting query $Q_{S\to T}$ given in Def. 7.1 and the formulae in the translation scheme. But when we compare the components carefully, we detect that there are similarities. The similarities are hidden behind the different concepts used by view support and translation schemes.

We start the comparison with the schema components. In Definition 7.1 we employ an ad-hoc schema transformation by describing how we change schema components. With translation schemes we can put these descriptions on solid grounds. The "$\mathcal{S}$-definition"[1] $\phi_{\mathrm{CLASS}}(C)$ binds a variable to the already existing set of classes by the first term in the disjunction. In Definition 7.1 this is expressed by placing the set of classes $\mathrm{CLASS}_S$ of the original schema into the union. The right side of the disjunction $C \overset{\circ}{=} p$ allows the variable $C$ to range over $p$ as well if the formula is true. The correspondence in the definition of $\mathrm{CLASS}_T$ is the singleton $\{p\}$ in the union.

The same correspondence can be found when we look at the "$\mathcal{S}$-definition" $\phi_{\mathrm{METH}}(M)$. The addition of new methods is captured in the disjunction $(M \overset{\circ}{=} c\_m_p\_m_1) \vee \cdots$. The removal of the obsolescent methods $m \in \mathbb{M}$ is embodied in the conjunction $M \overset{\circ}{\neq} m_1 \wedge \cdots$. The succession to the remaining methods is expressed by the factor $\mathrm{meth}(M)$.

The real $\mathcal{S}$-definition of :: follows the same principles. The same applies to the formula $\phi_{\Rightarrow}(M, C, R)$ only that the formulae become more complex. The first three lines in the defining formula $\phi_{\Rightarrow}$ correspond to the first line in the definition of $\mathrm{SIG}_T$ in Def. 7.1. They introduce the new signature declarations for the newly created attributes. These declarations are not completely independent from the signature declarations of the

[1]The formula $\phi_{\mathrm{CLASS}}(C)$ is not an $\mathcal{S}$-definition in the formal sense, but it defines the set of classes in the same spirit, hence this misuse of notation.

$$
\begin{aligned}
\phi_{\text{CLASS}}(C) &:= \text{class}(C) \lor C \overset{\circ}{=} p \\
\phi_{\text{METH}}(M) &:= (\text{meth}(M) \land M \overset{\circ}{\neq} m_1 \land \cdots \land M \overset{\circ}{\neq} m_n) \lor \\
&\qquad (M \overset{\circ}{=} c\_m_p\_m_1) \lor \cdots \lor (M \overset{\circ}{=} c\_m_p\_m_n) \\
\phi_{::}(C,D) &:= (C{::}D) \lor (C \overset{\circ}{=} p \land D \overset{\circ}{=} c_p) \\
\phi_{\Rightarrow}(M,C,R) &:= (M \overset{\circ}{=} c\_m_p\_m_1 \land C \overset{\circ}{=} p \land c[m_1 \Rightarrow R]) \\
&\qquad \lor \cdots \lor \\
&\qquad (M \overset{\circ}{=} c\_m_p\_m_n \land C \overset{\circ}{=} p \land c[m_n \Rightarrow R]) \lor \\
&\qquad \Big(C[M \Rightarrow R] \land \neg\big((M \overset{\circ}{=} m_1 \land C \overset{\circ}{=} c) \lor \\
&\qquad\qquad (M \overset{\circ}{=} m_2 \land C \overset{\circ}{=} c) \lor \\
&\qquad\qquad \vdots \\
&\qquad\qquad (M \overset{\circ}{=} m_n \land C \overset{\circ}{=} c) \lor \\
&\qquad\qquad (M \overset{\circ}{=} m_p \land C \overset{\circ}{=} c)\big)\Big) \lor \\
&\qquad (M \overset{\circ}{=} m_p \land C \overset{\circ}{=} c \land R \overset{\circ}{=} p) \\
\phi_{:}(O,C) &:= (O{:}C) \lor \big(C \overset{\circ}{=} p \land \exists O_c(O_c{:}c \land O_c[m_p \to O])\big) \\
\phi_{\to}(M,O,R) &:= (O[M \to R] \land M \overset{\circ}{\neq} m_1 \land \cdots \land M \overset{\circ}{\neq} m_n) \lor \\
&\qquad \big(M \overset{\circ}{=} c\_m_p\_m_1 \land \exists O_c(O_c[m_p \to O] \land O_c[m_1 \to R])\big) \\
&\qquad \lor \cdots \lor \\
&\qquad \big(M \overset{\circ}{=} c\_m_p\_m_n \land \exists O_c(O_c[m_p \to O] \land O_c[m_n \to R])\big)
\end{aligned}
$$

Table 7.1: A translation scheme for pivoting

obsolescent attributes. The new attributes take over the result classes of the respective old attributes. Then we carry over the attributes defined in the original schema except for the attributes in the set $\mathbb{M} \cup \{m_p\}$. Finally, we add the declaration of the signature $c[m_p \Rightarrow p]$.

The population of classes is done based on the old population as indicated in the left side of the disjunction of $\phi_{:}(O,C)$. The right side expresses that whenever an object $O$ is referenced by an object of class $c$ via attribute $m_p$, the object $O$ is a member of the pivot class $p$. The formula $\phi_{:}$ is reflected in Def. 7.1 in the supporting query $Q_{S\to T}$. We find the effect of $\phi_{:}$ at two places in $Q_{S\to T}$; once in the set $\{O : d \longleftarrow O : d \mid \cdots\}$ and in the set $\{P : p[c\_m_p\_m \to M] \longleftarrow \cdots\}$.

The $\mathcal{S}$-definition of $\to_0$ defines first the values for all old attributes without the pivoted attributes and then for all new attributes.

What remains is to show that we really have a translation scheme. So we have to check whether the restrictions in Def. 6.4 are met, but here we merely give an outline of the proof: The finiteness follows from the finiteness of the original schema and the independence from the extension from the syntactic form of the defining formulae.

Secondly, the left inverse $\Psi$ of the translation scheme $\Phi$ in Table 7.1 is defined in Table 7.2. The inverse nature becomes apparent when comparing the formulae one by one with the respective ones of the translation scheme $\Phi$.

$$\begin{aligned}
\psi_{\text{CLASS}}(C) &:= \text{class}(C) \wedge C \overset{\circ}{\neq} p \\
\psi_{\text{METH}}(M) &:= (\text{meth}(M) \wedge M \overset{\circ}{\neq} c\_m_p\_m_1 \wedge \cdots \wedge M \overset{\circ}{\neq} c\_m_p\_m_n) \vee \\
&\phantom{:=}\ (M \overset{\circ}{=} m_1) \vee \cdots \vee (M \overset{\circ}{=} m_n) \\
\psi_{::}(C, D) &:= C{::}D \wedge \neg(C \overset{\circ}{=} p \wedge D \overset{\circ}{=} c) \\
\psi_{\Rightarrow}(M, C, R) &:= (M \overset{\circ}{=} m_1 \wedge C \overset{\circ}{=} c \wedge p[c\_m_p\_m_1 \Rightarrow R]) \\
&\phantom{:=}\quad \vee \cdots \vee \\
&\phantom{:=}\ (M \overset{\circ}{=} m_n \wedge C \overset{\circ}{=} c \wedge p[c\_m_p\_m_n \Rightarrow R]) \vee \\
&\phantom{:=}\ (M \overset{\circ}{=} m_p \wedge C \overset{\circ}{=} c \wedge R \overset{\circ}{=} c_p) \vee \\
&\phantom{:=}\ \Big(C[M \Rightarrow R] \wedge \neg\big((M \overset{\circ}{=} m_p \wedge C \overset{\circ}{=} c \wedge R \overset{\circ}{=} p) \vee \\
&\phantom{:=}\qquad\qquad\qquad (M \overset{\circ}{=} c\_m_p\_m_1 \wedge C \overset{\circ}{=} p) \\
&\phantom{:=}\qquad\qquad\qquad\quad \vee \cdots \vee \\
&\phantom{:=}\qquad\qquad\qquad (M \overset{\circ}{=} c\_m_p\_m_n \wedge C \overset{\circ}{=} p)\big)\Big) \\
\psi_{:}(O, C) &:= O{:}C \wedge C \overset{\circ}{\neq} p \\
\psi_{\rightarrow}(M, O, R) &:= (O[M \rightarrow R] \wedge M \overset{\circ}{\neq} c\_m_p\_m_1 \wedge \cdots \wedge M \overset{\circ}{\neq} c\_m_p\_m_n) \vee \\
&\phantom{:=}\ \big(M \overset{\circ}{=} m_1 \wedge \exists P(O[m_p \rightarrow P] \wedge P[c\_m_p\_m_1 \rightarrow R])\big) \\
&\phantom{:=}\quad \vee \cdots \vee \\
&\phantom{:=}\ \big(M \overset{\circ}{=} m_n \wedge \exists P(O[m_p \rightarrow P] \wedge P[c\_m_p\_m_n \rightarrow R])\big)
\end{aligned}$$

Table 7.2: An inverse translation scheme for pivoting

Before we carry on our work in the notation of Biskup, we sketch the impact of the translation schemes $\Phi$ and $\Psi$ for pivoting on path functional dependencies. We start with a path functional dependency declared in the original schema $S$. The dependency is of the form

$$c(p_1 \cdots p_k \rightarrow p_{k+1} \cdots p_n) \ .$$

The corresponding path functional dependency F-formulae are

$$\begin{aligned} X[p \rightarrow P] \longleftarrow\ & X : c[p_1 \rightarrow P_1; \dots ; p_k \rightarrow P_k] \wedge \\ & Y : c[p_1 \rightarrow P_1; \dots ; p_k \rightarrow P_k; p \rightarrow P] \end{aligned}$$

for each $p \in \{p_{k+1}, \dots , p_n\}$. A term $X[p \rightarrow Y]$ where $p \equiv .m_1^p \cdots .m_l^p \in \mathrm{P}_{\mathrm{wf}}(D)\backslash\{.\mathrm{Id}\}$ is an abbreviation for

$$(\exists X_1^p \cdots \exists X_{l-1}^p (X[m_1^p \rightarrow X_1^p] \wedge X_1^p[m_2^p \rightarrow X_2^p] \wedge \cdots \wedge X_{l-1}^p[m_l^p \rightarrow Y])) \ .$$

What does pivoting make of any of these $X_i^p[m_{i+1}^p \rightarrow X_{i+1}^p]$? We can supply the answer to this question, when we explore $\Psi^{\#}(X_i^p[m_{i+1}^p \rightarrow X_{i+1}^p])$. The relevant part of $\Psi$ in this case is $\psi_{\rightarrow}(M, O, R)$. An analysis of $\psi_{\rightarrow}$ reveals that

$$\Psi^{\#}(X_i^p[m_{i+1}^p \rightarrow X_{i+1}^p]) = X_i^p[m_{i+1}^p \rightarrow X_{i+1}^p]$$

if $m_{i+1}^p \notin \mathbb{M}$ and

$$\Psi^{\#}(X_i^p[m_{i+1}^p \rightarrow X_{i+1}^p]) = \exists P(X_i^p[m_p \rightarrow P] \wedge P[c\_m_p\_m_{i+1}^p \rightarrow X_{i+1}^p])$$

if $m_{i+1}^p \in \mathbb{M}$. Therefore the application of $\Psi^{\#}$ on path functional dependency F-formulae yields path functional dependency F-formulae. In particular exactly those that are obtained, when pivoting is performed in the standard way.

## 7.4 Semantic Constraints under Pivoting

Apart from the redundancy removing effect of pivoting, we can observe the following behaviour. Looking at the semantic constraints exposed in Exam. 7.3 reveals that the path functional dependency

$$\mathsf{Phone\text{-}admin}(.\mathsf{fac} \rightarrow .\mathsf{fac.sch}) \tag{7.14}$$

is redundant as well. As already shown in Exam. 4.45, (7.14) can be derived from the trivial path functional dependency

$$\mathsf{Phoning\text{-}faculty}(.\mathrm{Id} \rightarrow .\mathsf{sch}) \ ,$$

which in turn can be derived from (7.14) and the onto constraint

$$\mathsf{Phone\text{-}admin}\{\mathsf{fac}|\mathsf{Phoning\text{-}faculty}\}$$

by "simple prefix reduction". In fact, each path functional dependency in the original schema of the form

$$\mathsf{Phone\text{-}admin}(.\mathsf{fac} \rightarrow .m)$$

with $m$ a pivoted attribute is transformed into a path functional dependency

$$\mathsf{Phone\text{-}admin}(.\mathsf{fac} \rightarrow .\mathsf{fac}.m) \ ,$$

which is redundant in the pivoted schema.

This behaviour is called in [BMPS96] *natural enforcement* of dependencies, because this kind of path functional dependencies is enforced due to a feature of object-oriented data models, namely by the fact that

$$\textsf{Phoning-faculty}(.\mathrm{Id} \rightarrow .m)$$

is a trivial path functional dependency in the pivoted schema. This natural enforcement can be compared with multi-valued dependencies that trivially hold in any unfolding of a nested relation scheme, but are not part of a nested relational schema.

The following theorem captures this behaviour formally. We show that we can remove a path functional dependency in the pivoted schema when the original path functional dependency is of the form

$$c(L \rightarrow R)$$

where $L$ comprises the pivot attribute $m_p$, $.m_p \in L$, and apart from that the sets $L$ and $R$ consist only of pivoted attributes, $(L\backslash\{.m_p\}) \cup R \subset \mathbb{M}$. In addition we can simplify some path functional dependencies in the pivoted schema, namely those whose corresponding path functions in the original schema contain only pivoted attributes as path functions. So if a path functional dependency $c(L \rightarrow R)$ with only pivoted attributes as path functions, $L \cup R \subset \mathbb{M}$, exists in the original schema, we substitute the path functional dependency $p(L \rightarrow R)$ for the corresponding path functional dependency in the pivoted schema.

Formally, this is done in the theorem by removing all path functional dependencies (the set (7.15)) whose corresponding original path functional dependencies are of the form $c(L \rightarrow R)$ with $L \subset \{.m_p\} \cup \mathbb{M}$ and $R \subset \mathbb{M}$ and then adding the set (7.16) of path functional dependencies of the form $p(L \rightarrow R)$ with $L \cup R \subset \mathbb{M}$.

To obtain from an original path functional dependency its pivoted counterpart, we introduce the operation $\mathcal{P}_{c,m_p}$. This operation performs the same substitution for pivoted attributes as was applied in Def. 7.1. The simplified forms are gained by means of the operation $\mathcal{R}_{c,m_p}$, which replaces a pivoted attribute by its "cryptic" counterpart.

We blur the clear distinction between attributes and path functions of length 1 in the following expositions.

Definition 7.5 (Simplification)
*Let $S$ be a schema, and the vector $(T, Q_{T\rightarrow S}, Q_{S\rightarrow T}) := \mathrm{piv}(S, p, c, \mathbb{M}, m_p)$ be the outcome of pivoting the schema $S$.*

- *The set $\mathcal{P}_{c,m_p}(M)$ for a set $M \subset \mathrm{Attr}_S(c)$ of attributes for the class $c$ is*

$$\begin{array}{lll} \{m' \mid m' = .m_p.c\_m_p\_m, & \textit{if} \quad m \in \mathbb{M} \cap M, & \textit{and} \\ \phantom{\{m' \mid{}} m' = .m & \textit{if} \quad m \in M\backslash\mathbb{M}\} & . \end{array}$$

- *The set $\mathcal{R}_{c,m_p}(M)$ for a set $M \subset \mathrm{Attr}_S(c)$ of attributes for the class $c$ is*

$$\{.c\_m_p\_m \mid m \in M\} \ .$$

- *The schema* $\mathrm{sim}(S, p, c, \mathbb{M}, m_p)$ *is defined*
  - *by removing path functional dependencies that stem from path functional dependencies of the form* $c(L \to R)$ *where* $\{.m_p\} \subset L \subset \{.m_p\} \cup \mathbb{M}$ *and* $R \subset \mathbb{M}$*, because these are naturally supported, and*
  - *by simplifying path functional dependencies that stem from path functional dependencies of the form* $c(L \to R)$ *where* $L \cup R \subset \mathbb{M}$*, because they can be simplified.*

$$\begin{aligned}
&\mathrm{sim}(S, p, c, \mathbb{M}, m_p) := \\
&\quad \Big(T \backslash \{c(\mathcal{P}_{c,m_p}(L) \to \mathcal{P}_{c,m_p}(R)) \mid c(L \to R) \in \mathrm{PFD}_S \textit{ and} \qquad (7.15)\\
&\qquad\qquad\qquad\qquad L \subset \{m_p\} \cup \mathbb{M} \textit{ and } R \subset \mathbb{M}\}\Big) \\
&\quad \cup \\
&\quad \{p(\mathcal{R}_{c,m_p}(L) \to \mathcal{R}_{c,m_p}(R)) \mid c(L \to R) \in \mathrm{PFD}_S \textit{ and } L \cup R \subset \mathbb{M}\} \quad (7.16)
\end{aligned}$$

Theorem 7.4 gave conditions under which the original schema and the pivoted schema were equivalent with respect to view instance support. As it turns out, these conditions are sufficient to ensure the equivalence between the pivoted schema and its simplifications.

Lemma 7.6 (Equivalence of Simplified Schemas)
*Let* $S$ *be a schema, and the vector* $(T, Q_{T\to S}, Q_{S\to T}) := \mathrm{piv}(S, p, c, \mathbb{M}, m_p)$ *be the outcome of pivoting the schema* $S$ *where* $\mathbb{M} \subset \{.m_p\}^{+_{S,c}}$*. Then the schema* $T$ *provides unique view instance support for the schema* $\mathrm{sim}(S, p, c, \mathbb{M}, m_p)$ *with supporting query* $\mathrm{id}_T$ *and vice versa.*

Proof. (7.15) and (7.16) merely change the semantic constraints of the schema $T$. We will show that despite these changes the semantic constraints of the schemas $T$ and $\mathrm{sim}(S, p, c, \mathbb{M}, m_p)$ are equivalent, and therefore the schemas are equivalent with respect to view instance support.

If $c(L \to R) \in \mathrm{PFD}_S$ is a path functional dependency with $\{.m_p\} \subset L \subset \{.m_p\} \cup \mathbb{M}$ and $R \subset \mathbb{M}$, then $c(\mathcal{P}_{c,m_p}(L) \to \mathcal{P}_{c,m_p}(R))$ is removed in the schema $\mathrm{sim}(S, p, c, \mathbb{M}, m_p)$. Now due to reflexivity, the path functional dependency $p(.\mathrm{Id} \cup \mathcal{R}_{c,m_p}(L \backslash \{.m_p\}) \to .\mathrm{Id})$ is trivially valid in the schema $\mathrm{sim}(S, p, c, \mathbb{M}, m_p)$ as well as, by simple attribution, the path functional dependency $p(.\mathrm{Id} \to \mathcal{R}_{c,m_p}(R))$. By simple prefix augmentation with the prefix $.m_p$, these dependencies imply $c(\mathcal{P}_{c,m_p}(L) \to .m_p)$ and $c(.m_p \to \mathcal{P}_{c,m_p}(R))$, respectively. By transitivity, we get $c(\mathcal{P}_{c,m_p}(L) \to \mathcal{P}_{c,m_p}(R))$, which was removed from the schema $T$ but still holds in the schema $\mathrm{sim}(S, p, c, \mathbb{M}, m_p)$.

If $c(L \to R) \in \mathrm{PFD}_S$ is a path functional dependency with $L \cup R \subset \mathbb{M}$, then $c(\mathcal{P}_{c,m_p}(L) \to \mathcal{P}_{c,m_p}(R))$ is in the schema $T$. Then due to the onto constraint $c\{m_p|p\} \in \mathrm{SC}_T$ added to the schema $T$ and simple prefix reduction, $p(\mathcal{R}_{c,m_p}(L) \to$

$\mathcal{R}_{c,m_p}(R)) \in \mathrm{SC}_T^{+_T}$ holds, which was added to schema $\mathrm{sim}(S, p, c, \mathbb{M}, m_p)$ for the path functional dependency $c(L \to R)$.

□

The lemma above shows that it is safe to remove some kind of constraints and to replace others by simpler forms. An interesting question is what happens if we have as input schema one with proper and purely functional dependencies, i. e. we have a *proper schema.*

Definition 7.7 (Proper Schema)
*Let $D$ be a schema. The schema $D$ is said to be* proper *:iff*

- *all onto constraints in* $\mathrm{OC}_D$ *are* proper*, i. e., if $c\{m|d\} \in \mathrm{OC}_D$ is an onto constraint, then the attribute $m$ is a proper attribute for class $c$.*
- *all path functional dependencies in* $\mathrm{PFD}_D$ *are* functional*, i. e., if $c(L \to R) \in \mathrm{PFD}_D$ is a path functional dependency, all path-functions in $L \cup R$ have length* 1*, and*
- *all functional dependencies in* $\mathrm{PFD}_D$ *are* proper*, i. e., if $c(L \to R) \in \mathrm{PFD}_D$ is a functional dependency, all attributes in $L \cup R$ are proper attributes for class $c$.*

When we have a proper schema, it is sufficient to know the proper functional dependencies defined for a class to determine all implied proper functional dependencies for that class as the next lemma shows. In this case even the inference rules reflexivity, augmentation and transitivity as known from the relational data model are complete. This means many results from the relational data model can be applied in this case.

Lemma 7.8 (Completeness of Proper Inference)
*Let $D$ be a proper schema. All derivable proper functional dependencies for a class can be derived using only the inference rules reflexivity, augmentation and transitivity, i. e. if the inclusion* $\mathrm{sat}(D) \subset \mathrm{sat}(D \cup \{c(X \to Y)\})$ *for a non-trivial proper functional dependency $c(X \to Y)$ over the schema $D$ holds, we can derive the proper functional dependency by only using the inference rules reflexivity, augmentation and transitivity.*

Proof. We conduct this proof by contraposition, i. e. we assume conversely that we cannot derive the non-trivial, proper functional dependency $c(X \to Y)$ by only using the inference rules reflexivity, augmentation and transitivity and show then that there exists an extension $f$ with $f \in \mathrm{sat}(D)$ and $f \notin \mathrm{sat}(D \cup \{c(X \to Y)\})$.

We define the *closure* $L^{*_{S,d}}$ for a set $L$ of proper attributes for a class $d$ in a schema $S$ as the set of proper attributes $l$ for the class $d$ where the proper functional dependency $c(L \to .l)$ can be derived by using the inference rules reflexivity, augmentation and transitivity.

We construct the extension $f$ of the schema $D$ satisfying the axioms AX and the semantic constraints $\mathrm{SC}_D$ of the schema $D$ but not the proper functional dependency $c(X \to Y)$. The extension $f$ consists of infinitely many objects $o, o_0, o_1, \ldots,$

but we are particularly interested in the objects $o$ and $o_0$, because they agree on their values for attributes in the closure $X^{*_{D,c}}$.

$$\begin{aligned} f := \langle\ & \{q{:}d \mid q \in \{o\} \cup \{o_i \mid i \in \mathbb{N}\} \text{ and } d \in \text{CLASS}_D\}| \\ & \{o[m \to o] \mid m \in \textstyle\bigcup_{d \in \text{CLASS}_D} \text{Attr}_D(d)\} \cup \{o_0[m \to o] \mid m \in X^{*_{D,c}}\} \cup \\ & \{o_{i+1}[m \to o_i] \mid i \in \mathbb{N} \text{ and } m \in X^{*_{D,c}}\} \cup \\ & \{o_i[m \to o_i] \mid i \in \mathbb{N} \text{ and } m \in \textstyle\bigcup_{d \in \text{CLASS}_D} \text{Attr}_D(d) \backslash X^{*_{D,c}}\}\ \rangle \end{aligned}$$

Fig. 7.3 renders how the extension $f$ looks like. An arrow signifies that for the object at the butt the values of the attributes in the label are the object at the point.

$$\begin{tikzcd} \cdots \arrow[r, "X^{*_{D,c}}"] & o_1 \arrow[loop above, "\bigcup_{d\in\text{CLASS}_D}\text{Attr}_D(d)\backslash X^{*_{D,c}}"] \arrow[r, "X^{*_{D,c}}"] & o_0 \arrow[loop above, "\bigcup_{d\in\text{CLASS}_D}\text{Attr}_D(d)\backslash X^{*_{D,c}}"] \arrow[r, "X^{*_{D,c}}"] & o \arrow[loop above, "\bigcup_{d\in\text{CLASS}_D}\text{Attr}_D(d)"] \end{tikzcd}$$

Figure 7.3: Sketch for the extension in the proof of Lem. 7.8

The extension $f$ satisfies the axioms AX and all onto constraints by definition.

We observe that if any two objects $q, r \in \text{pop}_f$ agree with their values for an attribute $m$, $\text{pop}_f \cup \text{ob}_f \models \exists O(q[m \to O] \wedge r[m \to O])$, then $\{q, r\} = \{o, o_0\}$ and $m \in X^{*_{D,c}}$ hold.

Let $c'(L \to R) \in \text{PFD}_D$ be a proper functional dependency for the class $c' \in \text{CLASS}_D \backslash \{c\}$. Then the set $L$ is a proper set of attributes for the class $c'$. Therefore $L \cap X^{*_{D,c}} = \emptyset$ holds, which entails that the proper functional dependency $c'(L \to R)$ is satisfied.

Let $c(L \to R) \in \text{PFD}_D$ be a proper functional dependency for the class $c$ and $q, r$ be two objects such that they agree on their values for all attributes in $L$. As observed above, $\{q, r\} = \{o, o_0\}$ and $L \subset X^{*_{D,c}}$ hold. Since $c(L \to R) \in \text{PFD}_D$ holds, $R \subset X^{*_{D,c}}$ is implied. Thus according to the construction of the extension $f$, the objects $q$ and $r$ agree on their values for attributes in the set $R$, which means the proper functional dependency $c(L \to R)$ is satisfied.

Therefore, the extension $f$ is an instance of the schema $D$, $f \in \text{sat}(D)$.

According to the construction of the extension $f$, the objects $o$ and $o_0$ agree on their values for attributes in the set $X$, but there exists an attribute $y \in Y \backslash X^{*_{D,c}}$ such that the objects $o$ and $o_0$ do not agree on their values for the attribute $y$. □

To achieve some of the results we are striving for, we have to normalise the syntactical form of proper functional dependencies further. We want their left-hand and right-hand sides to be as small as possible.

Definition 7.9 (L-minimum and Canonical)

- *A set of functional dependencies* $\Pi$ *over a proper schema* $D$ *is* L-minimum, *if for every* $c(X \to Y)$ *in* $\Pi$ *and for all* $\emptyset \neq X' \subsetneq X$ *we have* $D \cup \Pi \not\models c(X' \to Y)$*.*
- *A set of functional dependencies* $\Pi$ *over a proper schema* $D$ *is* canonical, *if the set* $\Pi$ *is L-minimum and the right-hand side of each dependency consists of just one path-function.*

A canonical set $\Pi$ equivalent to a set $\Pi'$ is a *canonical cover* of the set $\Pi'$. Such cover can be always found [MR92], when dealing with (traditional) functional dependencies. [MR92] contains algorithms to compute such a cover. Due to Lem. 7.8, we can use these algorithms in our setting, when we consider the proper functional dependencies for one class only.

When we have a proper schema with canonical functional dependencies, we can give necessary and sufficient conditions to ensure that the simplified pivoted schema is proper. These conditions can be found in a theorem in [BMPS96]. There the conditions were used to ensure the equivalence of the original and the pivoted schema. Since in our setting we have path functional dependencies at our disposal, the choice of the set of pivoted attributes $\mathbb{M} = \{.m_p\}^{*_{S,c}} \backslash \{.m_p\}$ is sufficient to ensure the equivalence.

Lemma 7.10 (Retaining Functional Dependencies)
*Let* $S$ *be a proper schema where* $\mathrm{PFD}_S$ *is canonical,* $T$ *be the schema* $\mathrm{sim}(S, p, c, \mathbb{M}, m_p)$ *where* $\mathbb{M} = \{.m_p\}^{*_{S,c}} \backslash \{.m_p\}$*. Then the following statements are equivalent:*

1. *for all subsets* $M \subset \mathrm{Pr}_S(c)$ *the conditions 1a, 1b and 1c hold:*
   (a) $M^{*_{S,c}} = (M \backslash \mathbb{M})^{*_{S,c}} \cup (M \cap \mathbb{M})^{*_{S,c}}$,
   (b) $(M \backslash \mathbb{M})^{*_{S,c}} \cap \mathbb{M} = \emptyset$ *or* $.m_p \in (M \backslash \mathbb{M})^{*_{S,c}}$*, and*
   (c) $(M \cap \mathbb{M})^{*_{S,c}} \cap (\mathrm{Pr}_S(c) \backslash \mathbb{M}) = \emptyset$*;*
2. *the schema* $T$ *is proper.*

Proof for 1 $\Longrightarrow$ 2. The set of functional dependencies $\mathrm{PFD}_S(c)$ in the schema $S$ can be partitioned into five sets.

$$\{c(L \to .r) \in \mathrm{PFD}_S \mid L \subset \mathrm{Pr}_S(c) \backslash \mathbb{M} \text{ and } r \in \mathrm{Pr}_S(c) \backslash \mathbb{M}\}\ , \tag{7.17}$$

$$\{c(L \to .r) \in \mathrm{PFD}_S \mid L \not\subset \mathrm{Pr}_S(c) \backslash \mathbb{M} \text{ and } r \in \mathrm{Pr}_S(c) \backslash \mathbb{M}\}\ , \tag{7.18}$$

$$\{c(L \to .r) \in \mathrm{PFD}_S \mid L \subset \mathbb{M} \text{ and } r \in \mathbb{M}\}\ , \tag{7.19}$$

$$\{c(L \to .r) \in \mathrm{PFD}_S \mid \{.m_p\} \neq L \not\subset \mathbb{M} \text{ and } r \in \mathbb{M}\} \text{ and} \tag{7.20}$$

$$\{c(L \to .r) \in \mathrm{PFD}_S \mid L = \{.m_p\} \text{ and } r \in \mathbb{M}\}\ . \tag{7.21}$$

We regard the sets (7.17) - (7.21) and show that the sets (7.18) and (7.20) are empty in the schema $S$.

So we consider a functional dependency $c(L \to .r) \in (7.18)$, which means $.r \in L^{*_{S,c}}$ holds. Therefore, due to condition 1a, $.r \in (L \backslash \mathbb{M})^{*_{S,c}}$ or $.r \in (L \cap \mathbb{M})^{*_{S,c}}$ must hold. By condition 1c, $.r \notin (L \cap \mathbb{M})^{*_{S,c}}$ can be implied, hence $.r \in (L \backslash \mathbb{M})^{*_{S,c}}$. This entails $c(L \backslash \mathbb{M} \rightarrow .r) \in \mathrm{SC}_D^{*_D}$, contrary to the fact that $\mathrm{PFD}_S$ is canonical.

Now we consider the functional dependency $c(L \rightarrow .r) \in (7.20)$. We distinguish two cases.

**$(L \backslash \mathbb{M})^{*_{S,c}} \cap \mathbb{M} = \emptyset$:** By condition 1a, $.r \in (L \cap \mathbb{M})^{*_{S,c}}$ follows. In this case, we consider two cases.

**$L \cap \mathbb{M} = \emptyset$:** Then, by definition, $(L \cap \mathbb{M})^{*_{S,c}} = \emptyset$ holds, in contradiction to condition 1a.

**$L \cap \mathbb{M} \neq \emptyset$:** Then $c(L \cap \mathbb{M} \rightarrow .r) \in \mathrm{PFD}_S(c)^{*_{S,c}}$ can be inferred. But since $L \cap \mathbb{M} \subsetneq L$ holds, the set $\mathrm{PFD}_S(c)$ is not canonical, in contradiction to the hypothesis.

**$(L \backslash \mathbb{M})^{*_{S,c}} \cap \mathbb{M} \neq \emptyset$:** By condition 1b, $.m_p \in (L \backslash \mathbb{M})^{*_{S,c}}$ holds, thereby $c(L \backslash \mathbb{M} \rightarrow .m_p) \in \mathrm{PFD}_S(c)^{*_{S,c}}$.

Because of $r \in \mathbb{M}$ and the choice of the set $\mathbb{M}$, $c(.m_p \rightarrow .r) \in \mathrm{PFD}_S(c)^{*_{S,c}}$ holds. By transitivity, $c(L \backslash \mathbb{M} \rightarrow .r) \in \mathrm{PFD}_S(c)^{*_{S,c}}$ holds, in contradiction to the fact that $\mathrm{PFD}_S$ is canonical.

Consequently, the functional dependencies of the schema $S$ are of type (7.17), (7.19) and (7.21).

Dependencies of type (7.17) are not changed by the transformation sim. A dependency $c(L \rightarrow .r)$ of type (7.19) is transformed into a proper functional dependency $p(\mathcal{R}_{c,m_p}(L) \rightarrow \mathcal{R}_{c,m_p}(\{.r\}))$. Dependencies of type (7.21) are removed by transformation sim. All onto constraints remain proper. Hence, the schema $T$ is proper.

Proof for 2 $\Longrightarrow$ 1. Since the schema $T$ is proper, all path functional dependencies are functional dependencies. Any functional dependency $c(L \rightarrow .r)$ of type (7.18) or (7.20) with $L \subsetneq \{.m_p\} \cup \mathbb{M}$ in the schema $S$ would have resulted in a non-functional dependency that would not have been removed by the transformation sim. A functional dependency $c(L \rightarrow .r)$ of type (7.20) with $L \subset \{.m_p\} \cup \mathbb{M}$ cannot exist because the set $\mathrm{PFD}_S(c)$ is canonical. Therefore all path functional dependencies in the schema $S$ must be of type (7.17), (7.19) or (7.21).

Due to Lem. 7.8, the closure $X^{*_{D,d}}$ of any set $X$ of proper attributes can be determined by exhaustively applying the inference rules reflexivity, augmentation and transitivity, i. e., whenever the left-hand sides of the antecedents are subsets of the so far calculated set, the right-hand sides are added to this set.

Since the dependencies in $\mathrm{PFD}_S(c)$ are either of type (7.17), (7.19) or (7.21), any derivation sequence can be ordered by first using all dependencies of type (7.17), then of type (7.21) and finally of type (7.19).

If we regard an attribute $a \in M^{*_{S,c}}$, then we can consider two cases.

1. The attribute $a$ occurs in the ordered derivation sequence after all applications of dependencies of type (7.17). In this case only attributes in $(M\backslash\mathbb{M})^{*_{S,c}}$ have been used as left-hand sides of dependencies, therefore $.a \in (M\backslash\mathbb{M})^{*_{S,c}}$.
2. The attribute $a$ does not occur in the ordered derivation sequence after all applications of dependencies of type (7.17). Again, we consider two cases.
   (a) The pivot attribute $.m_p$ occurs in the derivation sequence after all applications of dependencies of type (7.17). Then $.m_p \in (M\backslash\mathbb{M})^{*_{S,c}}$ and because of $\mathbb{M} \subset \{.m_p\}^{*_{S,c}}$, $.a \in (M\backslash\mathbb{M})^{*_{S,c}}$.
   (b) The pivot attribute $.m_p$ does not occur in the derivation sequence after all applications of dependencies of type (7.17). Then only attributes in $(M \cap \mathbb{M})^{*_{S,c}}$ have been used as left-hand sides of dependencies, therefore $.a \in (M \cap \mathbb{M})^{*_{S,c}}$.

Hence, condition 1a is satisfied by the schema $S$. Likewise we can show that conditions 1b and 1c are satisfied by the schema $S$.

□

## 7.5 Naturally Enforced Functional Dependencies

Our goal is to transform a schema such that all original functional dependencies are naturally enforced. In order to reach this objective, we have to consider two things. Firstly, it is in general impossible to discard all functional dependencies in one transformation step. Secondly, not all kinds of functional dependencies can be naturally enforced.

Example 7.11

*The first observation leads for example to a recursive application of the transformation as shown in Fig. 7.4 on the schema shown in Fig. 5.2.*

*We first choose* teacher *as pivot attribute with* room *and* wing *as pivoted attributes, which leads to the schema graph in Fig. 7.5. Then we perform pivoting on the resulting schema with pivot attribute* room *and pivoted attribute* wing *(Fig. 7.4).*

*We get the same outcome (Fig. 7.4) if we take first* room *as pivot attribute and* wing *as pivoted attribute (Fig. 7.6) and afterwards choose* teacher *as pivot attribute and* room *as only pivoted attribute. This example indicates that the outcome of pivoting is in a sense independent of the order in which single pivot steps are performed when we consider only transformations that lead to purely functional dependencies.*

*In this example we perform pivoting with transformation* sim. *So we remove some of the path functional dependencies. When we perform pivoting according to the instructions above, we obtain the schema in Fig. 7.7.*

*Note that we can discard all functional dependencies and that two onto constraints*

Teacher{room|Room}
Room{wing|Wing}

*wandered from the class* Assignment *to other classes.*

$$\begin{tikzcd}
 & \mathsf{Int} & & & & \\
\mathsf{Int} & \mathsf{Date} \arrow[l, "\mathsf{day}"'] \arrow[u, "\mathsf{month}"] \arrow[r, "\mathsf{year}"] & \mathsf{Int} & & & \\
\mathsf{String} & \mathsf{Course} \arrow[l, "\mathsf{title}"'] & \mathsf{Assignment} \arrow[l, "\mathsf{course}"'] \arrow[ul, "\mathsf{date}"] \arrow[dl, "\mathsf{assistant}"] \arrow[rr, "\mathsf{teacher}"] & & \mathsf{Teacher} \arrow[r, "\mathsf{te\_name}"] \arrow[d, "\mathsf{room}"] & \mathsf{String} \\
\mathsf{String} & \mathsf{Assistant} \arrow[l, "\mathsf{as\_name}"'] & & \mathsf{String} & \mathsf{Room} \arrow[l, "\mathsf{ro\_name}"'] \arrow[r, "\mathsf{size}"] \arrow[d, "\mathsf{wing}"] & \mathsf{Int} \\
 & & & & \mathsf{Wing} \arrow[r, "\mathsf{address}"] & \mathsf{String}
\end{tikzcd}$$

Figure 7.4: Applying recursive pivoting

$$\begin{tikzcd}
 & \mathsf{Int} & & & & \\
\mathsf{Int} & \mathsf{Date} \arrow[l, "\mathsf{day}"'] \arrow[u, "\mathsf{month}"] \arrow[r, "\mathsf{year}"] & \mathsf{Int} & & \mathsf{Wing} \arrow[r, "\mathsf{address}"] & \mathsf{String} \\
\mathsf{String} & \mathsf{Course} \arrow[l, "\mathsf{title}"'] & \mathsf{Assignment} \arrow[l, "\mathsf{course}"'] \arrow[ul, "\mathsf{date}"] \arrow[dl, "\mathsf{assistant}"] \arrow[rr, "\mathsf{teacher}"] & & \mathsf{Teacher} \arrow[r, "\mathsf{te\_name}"] \arrow[d, "\mathsf{room}"] \arrow[u, "\mathsf{wing}"] & \mathsf{String} \\
\mathsf{String} & \mathsf{Assistant} \arrow[l, "\mathsf{as\_name}"'] & & \mathsf{String} & \mathsf{Room} \arrow[l, "\mathsf{ro\_name}"'] \arrow[r, "\mathsf{size}"] & \mathsf{Int}
\end{tikzcd}$$

Figure 7.5: First applying pivoting with teacher as pivot attribute

*The schema graph for the schema above is shown in Fig. 7.8.*

*Noteworthy is that the two branches starting at the node representing the class* Assignment *with arrows labelled* course *and* teacher *are totally independent from each other. This independence is a result of the natural enforcement of all functional dependencies. Updates in one branch do not incur any look-ups or changes in the other branch.*

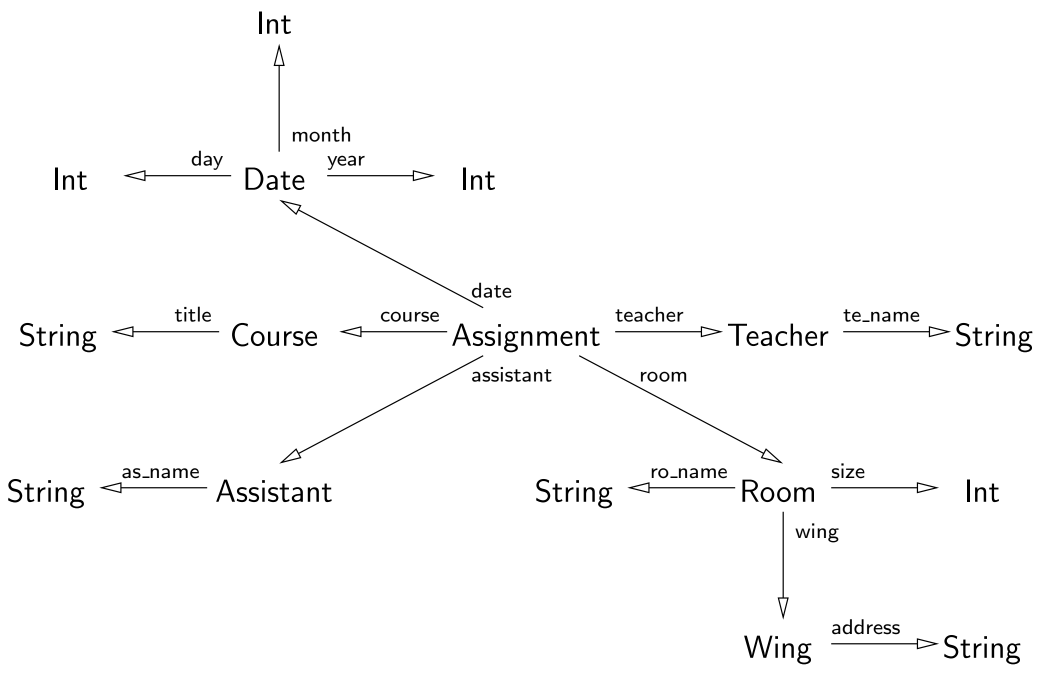


Figure 7.6: First applying pivoting with room as pivot attribute

*We can even dismiss the class* Assignment*. The class represents a binary relationship between* Course*s and* Teacher*s, which can be modelled by a set-valued method, which takes no arguments, either declared for the class* Course *or the class* Teacher*. The binary relationship is substantiated by the key path functional dependency*

$$\text{Assignment}(.\text{course} \quad .\text{teacher} \rightarrow .\text{Id}) \ .$$

*When we decide to model the relationship as a set-valued attribute for the class* Teacher*, we get the signature declaration*

$$\text{Teacher}[\text{course} \Rightarrow\!\!\!> \text{Course}]$$

*and can abolish the class* Assignment*. We do not pursue this aspect in this work, because general pivoting [BMP96] subsumes it.*

As said above not all kinds of functional dependencies can be naturally enforced. Natural enforcement of functional dependencies works only for those the left-hand side of which is a singleton, because pivoting can be applied only with at most one pivot attribute. Functional dependencies of this form are called unary functional dependencies [MR89]. We follow this notation and call path functional dependencies the left-hand side of which are singletons *unary functional dependencies.*

Definition 7.12 (Unary Path Functional Dependency)
*A path functional dependency $\pi$ is called a* unary *path functional dependency, if the left-hand side of the path functional dependency $\pi$ is a singleton.*

$$
\begin{aligned}
\mathrm{CLASS} &:= \{\mathsf{String}, \mathsf{Int}, \mathsf{Assignment}, \mathsf{Course}, \mathsf{Teacher}, \mathsf{Assistant}, \mathsf{Date}, \mathsf{Room}, \mathsf{Wing}\}\\
\mathrm{METH} &:= \{\mathsf{course}, \mathsf{teacher}, \mathsf{assistant}, \mathsf{room}, \mathsf{wing}, \mathsf{title}, \mathsf{te\_name}, \mathsf{as\_name}, \mathsf{ro\_name},\\
&\quad\ \mathsf{date}, \mathsf{year}, \mathsf{month}, \mathsf{day}, \mathsf{size}, \mathsf{address}\}\\
\mathrm{HIER} &:= \emptyset\\
\mathrm{SIG} &:= \{\mathsf{Course}[\mathsf{title} \Rightarrow \mathsf{String};\\
&\qquad\qquad \mathsf{assistant} \Rightarrow \mathsf{Assistant};\\
&\qquad\qquad \mathsf{date} \Rightarrow \mathsf{Date}],\\
&\quad\ \mathsf{Teacher}[\mathsf{te\_name} \Rightarrow \mathsf{String};\\
&\qquad\qquad \mathsf{room} \Rightarrow \mathsf{Room}],\\
&\quad\ \mathsf{Assistant}[\mathsf{as\_name} \Rightarrow \mathsf{String}],\\
&\quad\ \mathsf{Date}[\mathsf{year} \Rightarrow \mathsf{Int}; \mathsf{month} \Rightarrow \mathsf{Int}; \mathsf{day} \Rightarrow \mathsf{Int}],\\
&\quad\ \mathsf{Room}[\mathsf{size} \Rightarrow \mathsf{Int}; \mathsf{ro\_name} \Rightarrow \mathsf{String}; \mathsf{wing} \Rightarrow \mathsf{Wing}],\\
&\quad\ \mathsf{Wing}[\mathsf{address} \Rightarrow \mathsf{String}],\\
&\quad\ \mathsf{Assignment}[\mathsf{course} \Rightarrow \mathsf{Course};\\
&\qquad\qquad\quad \mathsf{teacher} \Rightarrow \mathsf{Teacher}]\}\\
\mathrm{SC} &:= \mathrm{AX} \cup\\
&\quad \{\mathsf{Assignment}(.\mathsf{course}\quad .\mathsf{teacher} \longrightarrow .\mathrm{Id}),\\
&\quad\ \mathsf{Assignment}\{\mathsf{teacher}|\mathsf{Teacher}\},\\
&\quad\ \mathsf{Assignment}\{\mathsf{course}|\mathsf{Course}\},\\
&\quad\ \mathsf{Teacher}\{\mathsf{room}|\mathsf{Room}\},\\
&\quad\ \mathsf{Room}\{\mathsf{wing}|\mathsf{Wing}\}\}
\end{aligned}
$$

Figure 7.7: Multiple pivoted schema

Unfortunately, the restriction to unary functional dependencies is not sufficient in order to eliminate all functional dependencies by recursive pivoting. To achieve this, we further have to make the set of pivoted attributes comprise all attributes in the closure of the pivot attribute in each transformation step. If we select as pivoted attributes all attributes in the closure of the pivot attribute without the pivot attribute itself, we call the underlying pivoting *maximal pivoting.*

Definition 7.13 (Maximal Pivoting)
*Pivoting a schema $D$ with* $\mathrm{sim}(D, p, c, \mathbb{M}, m_p)$ *is called* maximal *pivoting, if the set of pivoted attributes* $\mathbb{M}$ *comprises the closure of the pivot attribute except the pivot attribute itself,* $\mathbb{M} = \{.m_p\}^{*_{D,c}} \backslash \{.m_p\}$.

Now what thwarts maximal pivoting? The obstacle is a possible violation of the conditions given in Theor. 7.10. We will analyse these conditions in more depth.

Condition 1a is satisfied since we limit the use to unary functional dependencies. For having only sets of unary functional dependencies means that their closures are

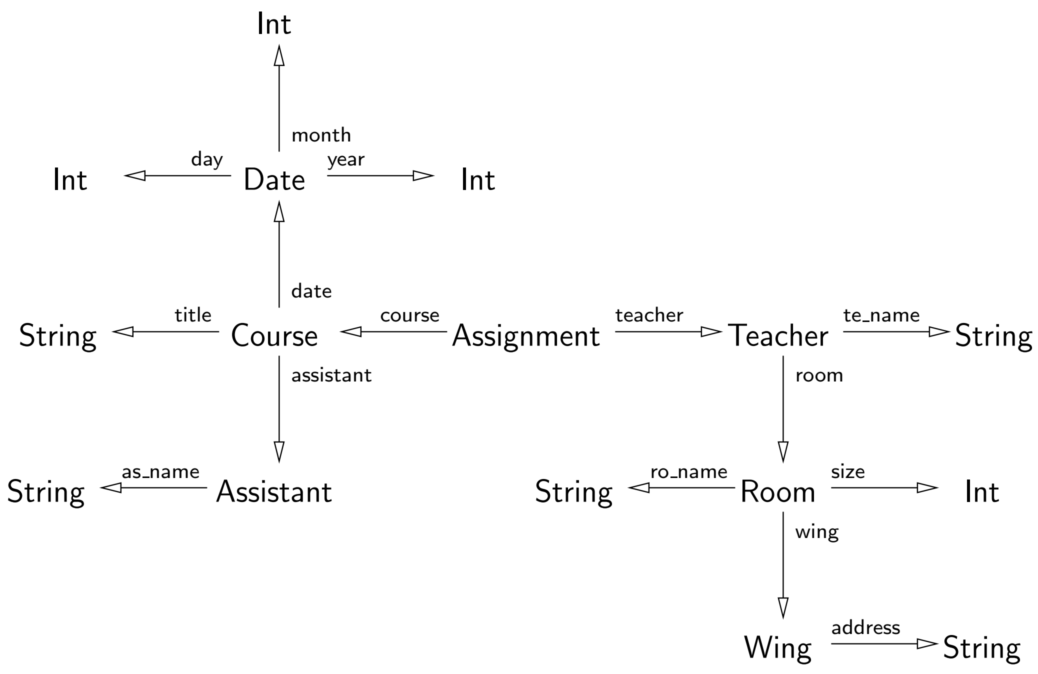


Figure 7.8: Applying recursive pivoting

topological [DLM92]. Therefore the equation

$$X^{*_{S,c}} = \bigcup_{a \in X} \{a\}^{*_{S,c}} \tag{7.22}$$

holds in a proper schema $S$, ensuing the fulfilment of condition 1a.

Condition 1b is harder to deal with. Here we consider a selection of a pivot attribute $m_p$ with a corresponding set $\mathbb{M}$ of pivoted attributes such that the selection violates condition 1b. This means there is a set of attributes or to be more precise due to (7.22) an attribute $m \in \mathrm{Attr}(c) \backslash \mathbb{M}$ such that $\{.m\}^{*_{S,c}} \cap \mathbb{M} \neq \emptyset$ and $.m_p \notin \{.m\}^{*_{S,c}}$. To describe this situation in a better way, we build a *dependency graph* for the set $\mathrm{PFD}(c)$ of functional dependencies.

The set of vertices of the graph is the set of attributes occurring in $\mathrm{PFD}(c)$. For each $L \to R_1 \cdots R_n \in \mathrm{PFD}(c)$ we add the edges $(L, R_i)$ to the graph. An example for this graph can be found in Fig. 7.9, which uses the functional dependencies in class Assignment in Fig. 5.1.

The graph describing the situation with the violation of condition 1b above is as depicted in Fig. 7.10. There is a path from $m$ to an attribute $m' \in \mathbb{M}$. Due to the fact that $m' \in \mathbb{M}$, there is a path from $m_p$ to $m'$. In addition there is neither a path from $m$ to $m_p$ nor vice versa. This kind of structure can be forbidden if we say that the graph has to form a forest, i. e., whenever there is one vertex reachable from two different nodes, one of these must be reachable by the other. Formally, we capture this without explicitly building the graph. *Subordination* expresses that any two attributes may not be simultaneously elements in their mutual closures.

$$\begin{tikzcd}
 & \text{course} \arrow[dl] \arrow[dr] & & \text{teacher} \arrow[d] \\
\text{assistant} & & \text{date} & \text{room} \arrow[d] \\
 & & & \text{wing}
\end{tikzcd}$$

Figure 7.9: A dependency graph

$$\begin{tikzcd}
m \arrow[rr, shift right, "\backslash" marking] \arrow[dr] & & m_p \arrow[ll, shift right, "\backslash" marking] \arrow[dl] \\
 & m' &
\end{tikzcd}$$

Figure 7.10: No forest

Definition 7.14 (Subordination)
*Let $D$ be a schema, $c \in \mathrm{CLASS}_D$ be a class, and $m, m' \in \mathrm{Pr}_D(c)$ be two proper attributes. The attributes $m$ and $m'$ are* subordinate *if either is an element of the closure of the other, $.m \in \{.m'\}^{*_{D,c}}$ or $.m' \in \{.m\}^{*_{D,c}}$ but not both.*

Definition 7.15 (Forest)
*Let $D$ be a proper schema, $c \in \mathrm{CLASS}_D$ be a class. The set $\mathrm{PFD}_D(c)$ of functional dependencies forms a* forest *if for all proper attributes $m, m' \in \mathrm{Pr}_D(c)$ for the class $c$, $m$ and $m'$ must be subordinate if their closures contain common elements, $\{.m\}^{*_{D,c}} \cap \{.m'\}^{*_{D,c}} \neq \emptyset$.*

Condition 1c is nearly satisfied since we take nearly all attributes in the closure as pivoted attributes. So condition 1c can only be violated if $.m_p \in \mathbb{M}^{*_{S,c}}$ holds. Which means since we are having only unary functional dependencies that $.m_p \in \{.m\}^{*_{S,c}}$ holds for some attribute $m \in \mathbb{M}$. Due to subordination condition 1c is satisfied, because if we require that any pairs of attributes in a schema are subordinate, $.m_p \notin \{.m\}^{*_{S,c}}$ holds for any attribute $m \in \mathbb{M}$.

We can give conditions under which we achieve a full natural enforcement for all functional dependencies of a class. One of the criteria is to use maximal sets of pivoted attributes, i. e. all attributes in the closure of a pivot attribute without the pivot attribute itself. We call recursive pivoting *recursive maximal pivoting* if at each stage of the transformation the set of pivoted attributes is maximal in the sense above. Finally, we get the following equivalence.

Theorem 7.16 (Full Natural Enforcement of Functional Dependencies)
*Let $D$ be a proper schema such that*

- $\mathrm{PFD}_D = \mathrm{PFD}_D(c)$ *for some class $c \in \mathrm{CLASS}_D$, and*
- $\mathrm{PFD}_D(c)$ *is canonical.*

*Then the following is equivalent:*

1. $\mathrm{PFD}_D(c)$ *is a set of unary functional dependencies and forms a forest.*

2. *Recursive maximal pivoting gives a proper schema without path functional dependencies.*

Proof for $1 \Longrightarrow 2$. Let $S_0, S_1, \ldots, S_s$ be a sequence of schemas performed for recursive maximal pivoting with $S_0 := D$ and $S_{i+1} = \mathrm{sim}(S_i, p_i, c_i, \mathbb{M}_i, m_{p_i})$.

By induction on the length $s$ of the transformation sequence, we show that

- each onto constraint in $\mathrm{OC}_{S_s}$ is proper, and
- for each class $d \in \mathrm{CLASS}_{S_s}$, the set of path functional dependencies $\mathrm{PFD}_{S_s}(d)$ is canonical and a set of proper, unary functional dependencies, and forms a forest.

$s = 0$**.** The schema $S_0 = D$ fulfils these properties by the hypotheses.

$s \to s+1$**.** Since $\mathrm{PFD}_{S_s}(c_s)$ is unary, the equation

$$X^{*S_s,c_s} = \bigcup_{a \in X} \{.a\}^{*S_s,c_s} \tag{7.23}$$

holds for any set of attributes $X \subset \mathrm{Pr}_{S_s}(c_s)$ [DLM92].

From (7.23) it immediately follows that condition 1a of Theor. 7.10 holds.

Let $M \subset \mathrm{Pr}_{S_s}(c_s)$ be a set of attributes. If an attribute $a'$ exists with $a' \in (M \backslash \mathbb{M}_s)^{*S_s,c_s} \cap \mathbb{M}_s$, then there exists an attribute $a \in M \backslash \mathbb{M}_s$ with $.a' \in \{.a\}^{*S_s,c_s}$ by (7.23). Since $\mathrm{PFD}_{S_s}(c_s)$ forms a forest and $\mathbb{M}_s = \{.m_{p_s}\}^{*S_s,c_s} \backslash \{.m_{p_s}\}$, the attributes $a$ and $m_{p_s}$ must be subordinate. So either $.a \in \{.m_{p_s}\}^{*S_s,c_s}$ or $.m_{p_s} \in \{.a\}^{*S_s,c_s}$ holds but not both. But since $a \in M \backslash \mathbb{M}_s$ holds, $.m_{p_s} \in \{.a\}^{*S_s,c_s}$ must hold, which means condition 1b of Theor. 7.10 is satisfied.

Let $M \subset \mathrm{Pr}_{S_s}(c_s)$ be a set of attributes. If an attribute $a'$ exists with $a' \in (M \cap \mathbb{M}_s)^{*S_s,c_s} \cap (\mathrm{Pr}_{S_s}(c_s) \backslash \mathbb{M}_s)$, then there exists an attribute $a \in M \cap \mathbb{M}_s$ with $.a' \in \{.a\}^{*S_s,c_s}$ by (7.23). Then due to the definition of $\mathbb{M}_s = \{.m_{p_s}\}^{*S_s,c_s} \backslash \{.m_{p_s}\}$, $a' = m_{p_s}$ and $.a \in \{.m_{p_s}\}^{*S_s,c_s}$ hold. Thus both $.m_{p_s} \in \{.a\}^{*S_s,c_s}$ and $.a \in \{.m_{p_s}\}^{*S_s,c_s}$ hold, in contradiction to the inductive assumption that $\mathrm{PFD}_{S_s}(c_s)$ forms a forest, which means condition 1c of Theor. 7.10 is satisfied.

Since conditions 1a, 1b and 1c of Theor. 7.10 are satisfied for the proper schema $S_s$ where the set $\mathrm{PFD}_{S_s}$ is canonical, the application of transformation sim produces, by Theor. 7.10, a proper schema $S_{s+1}$.

If either sets $\mathrm{PFD}_{S_{s+1}}(c_s)$ and $\mathrm{PFD}_{S_{s+1}}(p_s)$ were not canonical, the corresponding derivations could be lifted into the schema $S_s$. Thus the set $\mathrm{PFD}_{S_s}(c_s)$ would not be canonical.

Since $\mathrm{PFD}_{S_s}(c_s)$ contains only unary functional dependencies, the transformation sim does not introduce non-unary functional dependencies, hence the sets $\mathrm{PFD}_{S_{s+1}}(c_s)$ and $\mathrm{PFD}_{S_{s+1}}(p_s)$ contain only unary functional dependencies.

Because all functional dependencies are proper, the functional dependencies in $\mathrm{PFD}_{S_s}(d)$ do not contain any attribute in $\mathbb{M}_s$ for all classes $d \in \mathrm{CLASS}_{S_s} \backslash \{c_s\}$, and hence are not changed by the transformation sim. No path functional dependency is added to $\mathrm{PFD}_{S_{s+1}}(d)$. Therefore for each class $d \in \mathrm{CLASS}_{S_s} \backslash \{c_s\}$, the set of path functional dependencies $\mathrm{PFD}_{S_{s+1}}(d)$ remains canonical and a set of unary functional dependencies, and forms a forest.

Let $m, m' \in \mathrm{Pr}_{S_{s+1}}(c_s)$ be two attributes such that their closures are not disjunct, $\{.m\}^{*S_{s+1},c_s} \cap \{.m'\}^{*S_{s+1},c_s} \neq \emptyset$. We observe that removing functional dependencies from a set of functional dependencies makes its closure smaller. Therefore $\{.m\}^{*S_s,c_s} \cap \{.m'\}^{*S_s,c_s} \neq \emptyset$ holds. Since the set $\mathrm{PFD}_{S_s}(c_s)$ forms a forest, we assume w. l. o. g. $.m \in \{.m'\}^{*S_s,c_s}$. As was shown in the proof of Theor. 7.10 ($2 \Longrightarrow 1$) $.m \in \{.m'\}^{*S_s,c_s}$ can be derived by an ordered derivation sequence using only functional dependencies of type (7.17). Since these functional dependencies are not changed by the transformation sim, $.m \in \{.m'\}^{*S_{s+1},c_s}$ can be derived using the same sequence. Thus $\mathrm{PFD}_{S_{s+1}}(c_s)$ forms a forest.

The proof that $\mathrm{PFD}_{S_{s+1}}(p_s)$ forms a forest follows the same line of reasoning except that dependencies of type (7.19) are employed.

Now let $d(.m \rightarrow .m') \in \mathrm{PFD}_{S_0}(c_0)$ be a functional dependency. Due to the nature of the transformation sim and the properties of the schemas $S_i$ proven above, this functional dependency is either left unchanged, removed or altered into a different functional dependency in a transformation step. But the alteration just changes the class name and the attribute names. So this functional dependency can be traced throughout the transformation process. When the transformation uses the class $d$, possibly renamed, as pivot class and the attribute $m$, possibly renamed as well, as pivot attribute, then the attribute $m'$, again possible renamed, will be a pivoted attribute, since maximal pivoting is performed. Consequently, the functional dependency $d(.m \rightarrow .m')$ is removed in that very pivot step.

Proof for $2 \Longrightarrow 1$. Let $S_0, S_1, \ldots, S_s$ be a sequence of schemas performed for recursive maximal pivoting with $S_0$ be the final schema without path functional dependencies and $S_{i-1} = \mathrm{sim}(S_i, p_i, c_i, \mathbb{M}_i, m_{p_i})$.

We show that $\mathrm{PFD}_{S_s}(c)$ is a set of unary functional dependencies by contradiction. We assume there exists a functional dependency $c(L \rightarrow .r) \in \mathrm{PFD}_{S_s}(c)$ with $L = \{.m, .m'\}$. Since the schema $S_0$ does not contain any path functional dependencies, the functional dependency $c(L \rightarrow .r)$ or a variant of it must have been removed and not simplified in a maximal pivot step $i$. Hence, $\{.m_{p_i}\} \subset L \subset \{.m_{p_i}\} \cup \mathbb{M}$, and we assume w. l. o. g. $m_{p_i} \equiv m$. Then the functional dependency $c_i(.m \rightarrow .m') \in \mathrm{PFD}_{S_i}(c_i)^{*S_i,c_i}$ holds. A derivation sequence that witnesses this fact can be lifted into schema $S_s$ for the functional dependency $c(.m \rightarrow .m') \in \mathrm{PFD}_{S_s}(c)^{*S_s,c}$, in contradiction to the fact that $\mathrm{PFD}_{S_s}(c)$ is canonical.

Following the same line of reasoning, we can prove that every set $\mathrm{PFD}_{S_i}$ is canonical.

By induction on the length $s$ of the transformation sequence, we show that

- each onto constraint in $\mathrm{OC}_{S_s}$ is proper, and
- for each class $d \in \mathrm{CLASS}_{S_s}$, the set of path functional dependencies $\mathrm{PFD}_{S_s}(d)$ is a set of proper functional dependencies and forms a forest.

**$s = 0$.** The schema $S_0$ fulfils these properties by the hypotheses.

**$s \rightarrow s+1$.** Any onto constraint in $\mathrm{OC}_{S_s}$ is proper. Since the transformation sim does not make an onto constraint proper that has not been proper previously, any onto constraint in $\mathrm{OC}_{S_{s+1}}$ is proper.

The transformation sim adds only proper functional dependencies. Since all path functional dependencies in schema $S_s$ are proper, all path functional dependencies in the schema $S_{s+1}$ must have been proper.

Since the set $\mathrm{PFD}_{S_s}(d)$ for a class $d \in \mathrm{CLASS}_{S_{s+1}} \backslash \{c_s\}$ forms a forest and this set remains unchanged, the set $\mathrm{PFD}_{S_{s+1}}(d)$ forms a forest as well.

As was shown in the proof of Theor. 7.10 ($2 \Longrightarrow 1$), the set $\mathrm{PFD}_{S_{s+1}}(c_s)$ can be partitioned into the sets (7.17), (7.19) and (7.21).

We assume now for two attributes $m, m' \in \mathrm{Pr}_{S_{s+1}}(c_s)$ that their closures are not disjunct, $\{.m\}^{*S_{s+1},c_s} \cap \{.m'\}^{*S_{s+1},c_s} \neq \emptyset$, and distinguish four cases.

**$m, m' \in \mathrm{Pr}_{S_s}(c_s)$.** If $\{.m\}^{*S_{s+1},c_s} \cap \{.m'\}^{*S_{s+1},c_s} \cap \mathbb{M}_s \neq \emptyset$, then, by Theor. 7.10, $.m_p \in \{.m\}^{*S_{s+1},c_s} \cap \{.m'\}^{*S_{s+1},c_s}$. Therefore $.a \in \{.m\}^{*S_{s+1},c_s} \cap \{.m'\}^{*S_{s+1},c_s}$ exists and a derivation sequence of functional dependencies of type (7.17). Since these functional dependencies also exist in $\mathrm{PFD}_{S_s}(c_s)$, $.a \in \{.m\}^{*S_s,c_s} \cap \{.m'\}^{*S_s,c_s}$ holds. This means the attributes $m$ and $m'$ must be subordinate under the schema $S_s$. The corresponding derivation sequence can be reused under schema $S_{s+1}$.
If we assume $.m \in \{.m'\}^{*S_{s+1},c_s}$ and $.m' \in \{.m\}^{*S_{s+1},c_s}$, we can show, as above, that $.m \in \{.m'\}^{*S_s,c_s}$ and $.m' \in \{.m\}^{*S_s,c_s}$ holds, which contradicts the fact that $\mathrm{PFD}_{S_s}(c_s)$ forms a forest.

**$m, m' \in \mathbb{M}_s$.** Analogous with functional dependencies of type (7.19).

**$m \in \mathrm{Pr}_{S_s}(c_s)$, $m' \in \mathbb{M}_s$.** Due to condition 1c of Theor. 7.10, $\{.m'\}^{*S_{s+1},c_s} \subset \mathbb{M}_s$ holds, which entails $\{.m\}^{*S_{s+1},c_s} \cap \{.m'\}^{*S_{s+1},c_s} \subset \mathbb{M}_s$ and $.m \notin \{.m'\}^{*S_{s+1},c_s}$. By condition 1b of Theor. 7.10 and since maximal pivoting is performed, $.m' \in \{.m\}^{*S_{s+1},c_s}$ follows.

**$m \in \mathbb{M}_s$, $m' \in \mathrm{Pr}_{S_s}(c_s)$.** As above.

Hence, $\mathrm{PFD}_{S_{s+1}}(c_s)$ forms a forest.

□

The prerequisites in the theorem above appear to be very strict. But in fact we have to remember that for every set of functional dependencies there is a cover that is and canonical. Additionally, it is possible to relax the restrictions slightly. We propose two ways to do it without being formal.

The first relaxation is that we want to permit path functional dependencies that are of the form $c(A \rightarrow .\mathrm{Id})$. These are called *key path functional dependencies* [Wed92]. In doing so, we can handle the case where the sets of attributes in the closure of two pivot attributes are identical. If we include either of them into the set of pivot attributes of the other, let us say the pivot attribute $m_p'$ into the set of pivoted attributes $\mathbb{M}$ of pivot attribute $m_p$, we obtain $c(.m_p.c\_m_p\_m_p' \rightarrow .m_p)$ in the output schema. This path functional dependency is implied by $p(.c\_m_p\_m_p' \rightarrow .\mathrm{Id})$. Alternatively, if we really insist on functional dependencies, we tamper with the definition of maximal. We denote the set of attributes in the closure of the pivot attribute as maximal if it does not contain attributes that determine the pivot attribute functionally.

The next approach to relax the prerequisites benefits from a supplementary transformation. It is aimed at functional dependencies that cannot be expressed as unary functional dependencies. We assume that we have a functional dependency $c(A \rightarrow B)$, which is not unary, and no unary cover exists for it. We excise the set $A$ of attributes from class $c$ and introduce a new class $c_A$ with attributes $A$. We replace the set $A$ of attributes by an attribute $m_A$ in $c$. The path functional dependency $c(A \rightarrow B)$ is then mutated into $c(.m_A.A \rightarrow B)$. This path functional dependency is implied by $c(.m_A \rightarrow B)$ because $c_A(.\mathrm{Id} \rightarrow A)$ holds by successive application of "simple attribution". By "simple prefix augmentation", we can derive $c(.m_A \rightarrow .m_A.A)$. Due to transitivity, we get $c(.m_A \rightarrow B)$ as claimed.

## 7.6 Pivoting and Redundancy

In the introduction to this chapter we mention that a schema should possess nice properties. It is of importance in database design that theses properties find their way into rigorous definitions. If we fail to provide precise characterisations, we fail to prove that a schema possesses these properties. Mok et. al. [MNE96] show in their work what dangers arise, when a rigorous syntactic justification exists without a rigorous semantic justification. They carry out their work on database normalisation theory. They illustrate that when examples are provided instead of a rigorous semantic justification, these examples may be misleading.

As nice properties we mention in the introduction to this chapter that a schema should not structure redundant data. In order to give a rigorous semantic justification for this nice property, we pose the question, "What is redundant data?" Certainly, redundant data is not simply data that occurs literally at different locations in an instance. The fact that two persons have the same last name does not mean that their last names are redundant. But, when we know that both are husband and wife, it is very likely that they have the same last name. Let us put this observation more exactly. Data is redundant if we can reconstruct it from the remaining data and perhaps further knowledge over the application domain.

Our notation of redundancy is based on the idea that an attribute value is redundant if we can erase it, and from what remains and a single path functional dependency determine what the attribute value must have been before.

Definition 7.17 (Redundancy-Afflicted Attribute)
*Let $D$ be a schema, $c(X \rightarrow Y) \in (\mathrm{OC}_D \cup \mathrm{PFD}_D)^{+_D}$ be a path functional dependency such that $X, Y \subset \mathrm{Attr}_D(c)$ and $X \cap Y = \emptyset$. Let $f$ be an instance of $D$, $o, o' \in \mathrm{obj}(f)$ be two distinct objects, $\mathrm{compl}_D(f) \models o \overset{\circ}{\neq} o'$, of class $c$, $c \in \mathrm{l}_{\mathrm{Cl}}(o)$ and $c \in \mathrm{l}_{\mathrm{Cl}}(o')$. If the implication $\mathrm{compl}_D(f) \models o[m \rightarrow o_m] \wedge o'[m \rightarrow o_m]$ holds for all attributes $m \in X$, then the set of attributes $Y$ is* redundancy-afflicted *in the instance $f$ caused by the path functional dependency $c(X \rightarrow Y)$.*

Example 7.18
*In the instance $f$ of the schema in Fig. 5.3 with*

$$\mathrm{compl}_D(f) \models o[\mathsf{fac} \rightarrow \mathsf{bob}; \mathsf{sch} \rightarrow \mathsf{cs}; \mathsf{ph} \rightarrow 42; \mathsf{dep} \rightarrow \mathsf{engin}] \wedge \\ o'[\mathsf{fac} \rightarrow \mathsf{john}; \mathsf{sch} \rightarrow \mathsf{cs}; \mathsf{ph} \rightarrow 43; \mathsf{dep} \rightarrow \mathsf{engin}] \ .$$

*The attribute* dep *is redundancy-afflicted caused by the path functional dependency*

$$\mathsf{Phone\text{-}admin}(.\mathsf{sch} \rightarrow .\mathsf{dep}) \ .$$

The definition above is defined on the instance level. If a schema $D$ may have redundant instances, the schema has *potential redundancy.*

Definition 7.19 (Potential Redundancy)
*A schema $D$ is said to have* potential redundancy *if there exists a redundancy-afflicted set of attributes in any instance of $D$ caused by a path functional dependency implied by $\mathrm{OC}_D \cup \mathrm{PFD}_D$.*

Having a rigorous definition of redundancy, we continue by showing that pivoting reduces the redundancy potential of a schema. We conduct this examination on a rather informal level. We focus our attention on the instance in Exam. 7.18. We select the attribute sch in the left-hand side of the path functional dependency Phone-admin(.sch → .dep) as pivot attribute. The right-hand side is the candidate pivoted attribute. The output schema is sketched as follows:

$$\mathsf{Phone\text{-}admin}[\mathsf{fac} \Rightarrow \mathsf{Faculty}; \mathsf{sch} \Rightarrow \mathsf{School}; \mathsf{ph} \Rightarrow \mathsf{Phone}] \\ \mathsf{School}[\mathsf{dep} \Rightarrow \mathsf{Department}] \ .$$

We assume in this example that every School is associated with a Department. This association can only spring from the relationship modelled by the (relationship) class Phone-admin, and can be formalised as onto constraint Phone-admin{sch|School}. With this onto constraint the introduction of a subclass of the pivot class becomes superfluous.

Transforming the instance from Exam. 7.18 yields

$$\begin{aligned}\mathrm{compl}_D(f) \models\ & o[\mathsf{fac} \to \mathsf{bob}; \mathsf{sch} \to \mathsf{cs}; \mathsf{ph} \to 42] \wedge \\ & o'[\mathsf{fac} \to \mathsf{john}; \mathsf{sch} \to \mathsf{cs}; \mathsf{ph} \to 43] \wedge \\ & \mathsf{cs}[\mathsf{dep} \to \mathsf{engin}] \ .\end{aligned}$$

The redundancy-afflicted attribute dep is removed from class Phone-admin. So the redundancy potential with respect to the class Phone-admin is reduced. However, we do not increase the potential with respect to any other class. This reduction is no magical trick. It uses a feature of object-oriented data models, namely *sharing*. Two objects, in this case $o$ and $o'$, may reference the same value, cs. They share this value. Figure 7.11 gives an impression of how the situation looks like when we draw the instance. Objects are rectangles labelled with their oid. Their attributes are indicated by dividing the rectangles into drawers. The attribute name is written on the drawer and the arrow starting from it points to the attribute value. We contrast this digram with the diagram

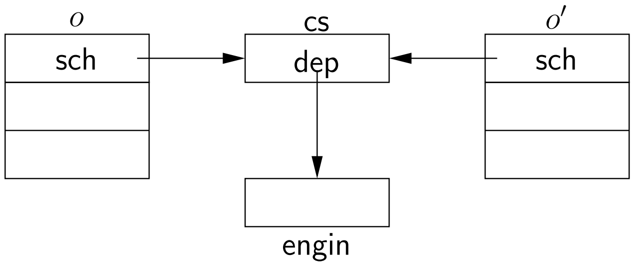


Figure 7.11: A redundancy-free, pivoted instance

representing the original situation before performing pivoting in Fig. 7.12. In this case both objects $o$ and $o'$ have a drawer for the attribute dep. Whereas the transformed instance has the attribute transplanted to the object cs.

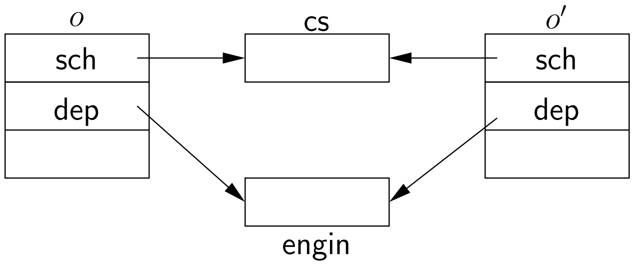


Figure 7.12: A “redundant” instance

The tight connection between redundancy and constraint enforcement becomes evident in the discussion above. By choosing the left-hand side of a path functional dependency causing redundancy, we can remove redundancy-afflicted attributes caused by this very path functional dependency. So the effect of pivoting is twofold. It decreases the redundancy potential and it increases the number of naturally enforced dependencies. Again we can make this point clearer by comparing Fig. 7.12 with Fig. 7.11. If the attribute value dep for object $o$ changes, we have to look up objects agreeing with object $o$ on their value for attribute sch. If none exists, we are done. But if we encounter such an object, we have a violation of the path functional dependency

$$\mathsf{Phone\text{-}admin}(.\mathsf{sch} \rightarrow .\mathsf{dep}) \ .$$

There are then several possibilities to deal with the situation

- rejecting the update,
- asking the originator of the update for her opinion,
- changing the value.

In the relational world this problem comes under the name *update anomalies.* There are normal forms for relational schemes to avoid these update anomalies and decompositions of relation schemas into these normal forms [MR92]. (Confer Sec. 7.7 which gives a broader perspective on this issue.) We can even make the observation that maximal recursive pivoting with the transformation sim on an input schema obeying the restrictions given in Theor. 7.16 leads to a schema without potential redundancy. This result is not surprising since all path functional dependencies are naturally enforced and onto constraints alone do not imply non-trivial path functional dependencies.

We do not conceal that these advantages have a price. The reduction of the redundancy potential and the easier enforcement of some path functional dependencies increase the costs for the enforcement of others. Pivoting changes the purely functional dependency

$$\mathsf{Phone\text{-}admin}(.\mathsf{ph} \rightarrow .\mathsf{dep})$$

into the path functional dependency

$$\mathsf{Phone\text{-}admin}(.\mathsf{ph} \rightarrow .\mathsf{sch}.\mathsf{dep}) \ ,$$

which cannot be further simplified, at least according to our current knowledge about pivoting. The attributes sch and dep are not comparable although their closures intersect. So there is a trade off between the removal of redundancy and natural enforcement of dependencies and the prolongation of path functions in path functional dependencies.

Now we disclose why we call Exam. 1.1 the “good” example and Exam. 1.2 the “bad” example. As we see in Fig. 7.9, the set of path functional dependencies for the class Assignment forms a forest. Therefore maximal recursive pivoting yields a schema as in Fig. 7.8 without any path functional dependency. Whereas the path functional dependencies for Exam. 1.2 do not form a forest. Performing pivoting with that example always comes with both:

- the removal of potential redundancy and natural enforcement of dependencies, and
- the prolongation of path functional dependencies.

## 7.7 Relational Normal Forms

Comparing pivoting with the restriction that we are confined to functional dependencies with the decomposition into Boyce-Codd normal form based on the work of Delobel and Casey [DC73], we find a strong resemblance between both transformations. This is not surprising as both transformations consider mainly sets of attributes and sets of functional dependencies. In fact we can even simulate pivoting in the relational model. Then it comes really close to the decomposition into Boyce-Codd normal form. In this case we use foreign keys in relations that represent relationship sets in order to access represented entities participating in a relationship. A subtle difference between both transformations is that pivoting uses object identificators for the reference mechanism whereas the relational model uses foreign keys, which are value oriented. Often in the modelling process using the relational model, keys are introduced that comprise only one attribute, e. g. a student number uniquely identifies a student. This can be seen as an attempt to simulate object identificators. Using this approach throughout the modelling process shifts pivoting even closer to the decomposition into Boyce-Codd normal form. Theorem 7.16 underlines the importance of unary functional dependencies as these dependencies can be naturally enforced. As a by-product, we know that if a set of functional dependencies consists only of unary functional dependencies, the corresponding *Armstrong relation* can be found in polynomial time [MR89].

When we allow the introduction of path functional dependencies in the output schemas, we can transform arbitrary schemas and still obtain an information-preserving and dependency-preserving transformation. In this respect pivoting is closed for path functional dependencies. This is in contrast to Boyce-Codd normal form. We cannot guarantee the preservation of dependencies, because the set of functional dependencies is not closed under the decomposition into Boyce-Codd normal form. Makowsky and Ravve [MR98] show there exist a translation refinement for a schema such that the following holds.

- The semantic constraints of the translation refinement comprises functional dependencies and inclusion dependencies.
- The translation schemes $\Phi$ and $\Psi$ that constitute the translation refinement are compositions of projections and joins, but the left inverse $\Psi$ uses *vectorisation.*
- The translation refinement is in Boyce-Codd normal form with respect to its implied functional dependencies.

Vectorisation becomes necessary, because they propose a solution by *splitting attributes.*

Looking at the dependency graph spanned by a set of functional dependencies in the case when it forms a forest, we see that this forest is a set of scheme trees. For a scheme tree $T$, MVD($T$) denotes the union of all the multi-valued dependencies represented by the edges in $T$ [MNE96]. Let $F$ be a set of functional dependencies that corresponds to a tree $T$ in a forest, i. e. a maximal connected subgraph. The tree $T$ is also a scheme tree and the set of multi-valued dependencies implied by $F$ is equal to the set of multi-valued dependencies MVD($T$). This result is obtained due to the definitions of a tree $T$ and the set MVD($T$). Thus we establish the relationship between Vincent et. al. [VS93] and Mok et. al. [MNE96], which investigate redundancies and normal forms.

Johnson and Fernandez [JF97] show that the normalisation process from relational database design remains useful in its object-oriented counterpart. They examine how relational normal forms including 3NF and BCNF can be used in re-engineering a relational schema into an object-oriented schema. Their work is carried out on an informal level. For instance, they bring an example that violates 3NF, and its synthesis into 3NF and an object-oriented schema reflecting this decomposition. But this object-oriented schema can be obtained using pivoting. In essence, they show how relationships that are broken into smaller components due to normal forms in the relational framework can be represented in an object-oriented framework.

# Chapter 8

# Conclusion

In this work we have investigated how to represent relationships in object-oriented data models. Our aim has been to put these investigations onto solid grounds, and our starting point for these investigations have been application-dependent semantic constraints, namely *class inclusion constraints*, *onto constraints* and *path functional dependencies*. In order to accomplish this task, we have devised a formal object-oriented data model that incorporates those semantic constraints.

We have analysed different representations of the same relationship with respect to constraint enforcement and redundancy. To obviate problems, we have delivered rigorous definitions for both constraint enforcement and redundancy. Because if we lack rigorous definitions, we fail to prove that schemas possess these properties.

We have obtained different representations by means of the database transformation *pivoting*, which breaks relationships represented as (relationship) classes into smaller components. The starting point for pivoting is a canonical representation of the relationship, which is later on decomposed. Instead of creating completely new classes as it is done in the relational data model, these components become subclasses of already existing classes.

Pivoting realises two principles: *abstraction* and *sharing*. Abstraction [VdB93] means the identification of information that can represent concepts in its own right, independent of its surroundings. To put the abstracted information to use, we extract the information from all sources and merge it. Finally, we establish references to the extracted information in the sources, i. e., the sources share the common information afterwards.

As 3NF and BCNF and their respective decompositions have found their way into practise, we conjecture that it is possible to devise a normal form for object-oriented schemas and use pivoting as decomposition. Accordingly, these results may find their way into practise as well, providing guidance in object-oriented database design. The closeness of pivoting to 3NF and BCNF is especially interesting in the field of reverse engineering of relational schemas. The potential normal form we have in mind is based on Theor. 7.16, but the details lay beyond the ken of this work.

To pave the path for pivoting to find its way into practise, we need the capability to syntactically manipulate semantic constraints. An issue which we have partly solved for class inclusion constraints, onto constraints and path functional dependencies. For the combination of these constraints, we have presented a sound and complete axiomatisation for our object-oriented data model, which includes inheritance and multiple class membership. However, the decision problem of the implication is still an open problem for the combination of class inclusion constraints, onto constraints and path functional dependencies. The problem is solved positively for the case where only path functional dependencies are allowed in a semantic data model [IW94, vBW94]. We surmise that when all onto constraints are acyclic, the decision procedure of Ito and Weddell can be used.

As our investigations have been based on different representations of the same relationship, the issue of whether these representations are really equivalent was part of the problem. To solve this problem, i. e., to compare the information capacity of an original schema and its transformed schemas, we have introduced *translation schemes* in our object-oriented data model. Translation schemes enable us to compare even the semantic constraints of schemas. These semantic constraints may stretch over formulae that deal with schema aspects, a capability not found in the relational data model. The application fields for translation schemes can be all areas of schema integration and translation [MIR93]. In particular, translation schemes offer a way to mediate between the relational data model and our object-oriented data model. This is because our definition of an object-oriented data model encompasses predicates, i. e. relations as first class citizens.

Still an open point is an investigation into the trade off between natural enforcement of semantic constraints and the prolongation of path functions in path functional dependencies. A cost model, in which we can measure the enforcement and update costs, is indispensable at this point.

# Bibliography

[ABD$^+$89] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, and S. Zdonik. The object-oriented database system manifesto. In Kim et al. [KNN89], pages 40–57. also: SIGMOD May 1990.

[ABGO93] A. Albano, R. Bergamini, G. Ghelli, and R. Orsini. An object data model with roles. In Agrawal [Agr93], pages 39–51.

[Abi90] S. Abiteboul. Towards a deductive object-oriented database language. *Data & Knowledge Engineering*, 5(2):263–287, 1990.

[ABLV83] P. Atzeni, C. Batini, M. Lezerini, and F. Villanelli. INCOD: A system for conceptual design of data and transactions in the entity-relationship model. In C. G. Davis, S. Jajodia, P. A.-B. Ng, and R. T. Yeh, editors, *Proceedings of the 3rd International Conference on Entity-Relationship Approach*, pages 375–410. Elsevier Science Publishers B. V (North-Holland), 1983.

[ABM80] G. Ausiello, C. Batini, and M. Moscarini. On the equivalence among database schemata. In S. Misbah Deen and P. Hammersley, editors, *Proceedings of the International Conference on Data Bases*, pages 34–46, University of Aberdeen, 1980. Heyden & Son.

[Agr93] R. Agrawal, editor. *Proceedings of the 19th International Conference on Very Large Data Bases*, Dublin, Irland, 1993.

[AH88] S. Abiteboul and R. Hull. Restructuring hierarchical database objects. *Theoretical Computer Science*, 62:3–38, 1988.

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, Reading, MA, 1995.

[AK89] S. Abiteboul and P. C. Kanellakis. Object identity as a query language primitive. In Clifford et al. [CLM89], pages 159–173.

[AK92] S. Abiteboul and P. C. Kanellakis. *Building an Object-Oriented Database System: The Story of $O_2$*, chapter 5 Object Identity as a Query Language Primitive. Morgan Kaufmann, 2929 Campus Drive, Suite 260, San Mateo, CA 94403, first edition, 1992.

[ALUW93] S. Abiteboul, G. Lausen, H. Uphoff, and E. Waller. Methods and rules. In P. Buneman and S. Jajodia, editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 32–41, 1993.

[AVdB95] S. Abiteboul and J. Van den Bussche. Deep equality revisited. In T. W. Ling, A. O. Mendelzon, and L. Vieille, editors, *Proceedings of the 4th Deductive and Object-Oriented Databases (DOOD '95)*, volume 1013 of *Lecture Notes in Computer Science*, pages 213–228, Singapore, 1995. Springer-Verlag, Berlin.

[Ban88] F. Bancilhon. Object-oriented database systems. In *Proceedings of the Seventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 152–162, 1988.

[Bar95] M. Bari. From conceptual specification to object-oriented design. In D. Patel, editor, *1994 International Conference on Object Oriented Information Systems*, pages 364–377. Springer-Verlag, 1995.

[Bat88] C. Batini, editor. *Proceedings of the 7th International Conference on Entity-Relationship Approach*, Rome, Italy, November 1988. Elsevier Science Publishers B.V (North-Holland).

[BB93] R. Bal and H. Balsters. A deductive and typed object-oriented language. In S. Ceri, T. Tanaka, and S. Tsur, editors, *Proceedings of the 3rd Deductive and Object-Oriented Databases (DOOD '93)*, volume 760 of *Lecture Notes in Computer Science*, pages 340–359, Phoenix, Arizona, USA, 1993. Springer-Verlag, Berlin.

[Bee89] C. Beeri. Formal models for object-oriented databases. In Kim et al. [KNN89], pages 405–430.

[BFP$^+$95] M. L. Barja, A. A. A. Fernandes, N. W. Paton, M. H. Williams, A. Dinn, and A. I. Abdelmoty. Design and implementation of ROCK & ROLL: a deductive object-oriented database system. *Information Systems*, 20(3):185–211, 1995.

[Bis95a] J. Biskup. Database schema design theory: Achievements and challenges. In S. Bhalla, editor, *Proceedings of the 6th International Conference Information Systems and Management of Data*, number 1006 in Lecture Notes in Computer Science, pages 14–44, Bombay, 1995. Springer-Verlag.

[Bis95b] J. Biskup. *Grundlagen von Informationssystemen.* Vieweg, Braunschweig-Wiesbaden, 1995.

[BLN86] C. Batini, M. Lenzerini, and S. B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323–364, 1986.

[BM92] E. Bertino and D. Montesi. Towards a logical object-oriented programming language for databases. In A. Pirotte, C. Delobel, and G. Gottlob, editors, *Advances in Database Technology—EDBT '92*, number 580 in Lecture Notes in Computer Science, pages 168–183. Springer-Verlag, 1992.

[BMP96] J. Biskup, R. Menzel, and T. Polle. Transforming an entity-relationship schema into object-oriented database schemas. In J. Eder and L. A. Kalinichenko, editors, *Advances in Databases and Information Systems, Moscow 95*, Workshops in Computing, pages 109–136. Springer-Verlag, 1996.

[BMPS96] J. Biskup, R. Menzel, T. Polle, and Y. Sagiv. Decomposition of relationships through pivoting. In Thalheim [Tha96], pages 28–41.

[BR88] J. Biskup and U. Räsch. The equivalence problem for relational database schemes. In J. Biskup, J. Demetrovics, J. Paredaens, and B. Thalheim, editors, *Proceedings of the 1st Symposium Mathematical Fundamentals of Database Systems*, number 305 in Lecture Notes in Computer Science, pages 42–70. Springer-Verlag, 1988.

[BT98] C. Beeri and B. Thalheim. Identification as a primitive of database models. In T. Polle, T. Ripke, and K.-D. Schewe, editors, *Proceedings of the Workshop on Foundations of Models and Languages for Data and Objects*, Timmel, Germany, 1998. Kluwer Publisher, New York.

[CC89] Q. Chen and W. W. Chu. A high-order logic programming language (HiLog) for non-1NF deductive databases. In Kim et al. [KNN89], pages 396–418.

[CCCR+90] F. Cacace, S. Ceri, S. Crespi-Reghizzi, L. Tanca, and R. Zicari. Integrating object-oriented data modeling with a rule-based programming paradigm. In H. Garcia-Molina and H. V. Jagadish, editors, *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, pages 225–236, 1990.

[Che76] P. P.-S. Chen. The entity-relationship-model — towards a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, March 1976.

[CK95] W. Chen and M. Kifer. Sorted HiLog: Sorts in higher-order logic data languages. In G. Gottlob and M. Y. Vardi, editors, *Database Theory — ICDT '95*, number 893 in Lecture Notes in Computer Science, Prague, 1995. Springer-Verlag.

[CL80] E. P. F. Chan and F. H. Lochovsky. A graphical data base design aid using the entity-relationship model. In P. P.-S. Chen, editor, *Proceedings of the International Conference on Entity-Relationship Approach*, pages 295–310. 1980.

[CLM89] J. Clifford, B. G. Lindsay, and D. Maier, editors. *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, 1989.

[CM90] A. F. Cárdenas and D. McLeod, editors. *Research Foundations in Object-Oriented and Semantic Database Systems.* Series in Data and Knowledge Base Systems. Englewood Cliffs, 1990.

[Cod72] E. F. Codd. Further normalization of the database relational model. In R. Rustin, editor, *Database Systems*, number 6 in Courant Institute Computer Science Symposia Series, pages 33–64. Prentice Hall, Englewood Cliffs, NJ, 1972.

[CW89] W. Chen and D. S. Warren. C-logic for complex objects. In PODS89 [POD89], pages 369–378.

[Dat94] C. J. Date. *An Introduction to Database Systems.* Addison-Wesley, sixth edition, 1994.

[dBL93] G. di Battista and M. Lenzerini. Deductive entity relationship modeling. *IEEE Transaction on Knowledge and Data Engineering*, 5(3):439–450, 1993.

[DC73] C. Delobel and R. G. Casey. Decomposition of a data base and the theory of boolean switching functions. *IBM Journal of Research and Development*, 17(5):374–386, 1973.

[DLM92] J. Demetrovics, L. O. Libkin, and I. B. Muchnik. Functional dependencies in relational databases: a lattice point of view. *Discrete Applied Mathematics*, 40:155–185, 1992.

[dSAD94] C. S. dos Santos, S. Abiteboul, and C. Delobel. Virtual schemas and bases. In M. Jarke, J. Bubenko, and K. Jeffery, editors, *Advances in Database Technology—EDBT '94*, number 779 in Lecture Notes in Computer Science, pages 81–94. Springer-Verlag, Berlin etc., 1994.

[DT93] O. De Troyer. *On Data Schema Transformation.* PhD thesis, Katholieke Universiteit Brabant, Netherlands, 1993. Wibro Dissertatiedrukkerij, Helmond, The Netherlands.

[DT95] G. Dobbie and R. W. Topor. On the declarative and procedural semantics of deductive object-oriented systems. *Journal of Intelligent Information Systems*, 4(2):193–219, 1995.

[DTM95] O. De Troyer and R. Meersman. A logic framework for a semantics of object oriented data modelling. In Papazoglou [Pap95], pages 238–249.

[EFT92] H.-D. Ebbinghaus, J. Flum, and W. Thomas. *Einführung in die mathematische Logik.* BI-Wiss.-Verl., Mannheim [u.a.], 3. vollst. überarb. und erw. Aufl. edition, 1992.

[EKT93] R. A. Elmasri, V. Kouramajian, and B. Thalheim, editors. *Proceedings of the 12th International Conference on Entity-Relationship Approach*, Arlington, Texas, USA, 1993.

[EN94] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems.* Benjamin/Cummings, Redwood City, CA, second edition, 1994.

[FV95] C. Fahrner and G. Vossen. A survey of database design transformations based on the Entity-Relationship model. *Data & Knowledge Engineering*, 15(1995):212–250, 1995.

[GC91] K. Gorman and J. Choobineh. The object-oriented entity-relationship model (OOERM). *Journal of Management Information Systems*, 7(3):41–65, 1991.

[Get92] J. R. Getta. Translation of extended entity-relationship database model into object-oriented database model. In D. K. Hsiao, E. J. Neuhold, and R. Sacks-Davis, editors, *Proceedings of the IFIP WG 2.6 Database Semantics Conference on Interoperable Database Systems*, pages 87–100, Lorne, Victoria, Australia, 1992. North-Holland 1993.

[GHC$^+$93] M. Gogolla, R. Herzig, S. Conrad, G. Denker, and N. Vlachantonis. Integrating the ER approach in an OO environment. In Elmasri et al. [EKT93], pages 376–389.

[GM92] J. Grant and J. Minker. The impact of logic programming on databases. *Communications of the ACM*, 35(3), 1992.

[Gog94] M. Gogolla. *An Extended Entity-Relationship Model.* LNiCS 767. Springer-Verlag, Heidelberg, 1994.

[GPvG90] M. Gyssens, J. Paradaens, and D. van Gucht. A graph-oriented object database model. In *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 417–424, Nashville, Tennessee, 1990.

[HB69] D. Hilbert and P. Bernays. *Grundlagen der Mathematik I.* Springer-Verlag, Berlin, 1969.

[HB70] D. Hilbert and P. Bernays. *Grundlagen der Mathematik II.* Springer-Verlag, Berlin, Zweite Auflage edition, 1970.

[HG92] R. Herzig and M. Gogolla. Transforming conceptual data models into an object model. In G. Pernul and A. M. Tjoa, editors, *Proceedings of the 11th International Conference on Entity-Relationship Approach*, number 645 in Lecture Notes in Computer Science, pages 280–298, Karlsruhe, Germany, 1992. Springer-Verlag.

[HMPR93] C. A. Heuser, E. Meira Peres, and G. Richter. Towards a complete conceptual model: Petri nets and entity-relationship diagrams. *Information Systems*, 18(5):275–298, 1993.

[Hoh93] U. Hohenstein. *Formale Semantik eines erweiterten Entity-Relationship-Modells.* Teubner, Stuttgart, 1993.

[Hul86] R. Hull. Relative information capacity of simple relational database schemata. *SIAM Journal on Computing*, 15(3):856–886, 1986.

[IW94] M. Ito and G. E. Weddell. Implication problems for functional constraints on databases supporting complex objects. *Journal of Computer and System Sciences*, 49(3):726–768, 1994.

[JF97] J. L. Johnson and G. Fernandez. Re-engineering relational normal forms in an object-oriented framework. In M. E. Orlowska and R. Zicari, editors, *1997 International Conference on Object Oriented Information Systems*, pages 433–452, Brisbane, 1997. Berlin.

[JOS93] P. Jaeschke, A. Oberweis, and W. Stucky. Extending ER model clustering by relationship clustering. Technical Report 273, Universität Karlsruhe, Institut für Angewandte Informatik und Formale Beschreibungsverfahren, 1993.

[Kim95] W. Kim. Editorial directions. *ACM Transactions on Database Systems*, 20(3):237–238, 1995.

[KL89a] M. Kifer and G. Lausen. F-logic: a higher order language for reasoning about objects, inheritance, and schema. In Clifford et al. [CLM89], pages 134–146.

[KL89b] W. Kim and F. H. Lochovsky, editors. *Object-Oriented Concepts, Databases, and Applications.* Frontier Series. ACM Press, New York, September 1989.

[KLR88] D. E. Knuth, T. Larrabee, and R. M. Roberts. Mathematical writing. Technical Report STAN-CS-88-1193, Department of Computer Science, Stanford University, Stanford, CA 94305, 1988.

[KLW95] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42(4):741–843, 1995.

[KNN89] W. Kim, J.-M. Nicolas, and S. Nishio, editors. *Proceedings of the 1st Deductive and Object-Oriented Databases (DOOD '89)*, Kyoto, Japan, 1989. Elsevier Science Publishers (North-Holland).

[Koh96] N. Kohlrausch. Äquivalenz von objekt-orientierten Datenbankschemas. Diplomarbeit, Institut für Informatik, Universität Hildesheim, 1996.

[KR97] H.-J. Klein and J. Rasch. Functional dependencies for object databases: Motivation and axiomatization. Technical Report 9706, Institut für Informatik und Praktische Mathematik, Universität Kiel, 1997.

[KS88] G. Kappel and M. Schrefl. A behavior integrated entity-relationship approach for the design of object-oriented databases. In Batini [Bat88], pages 311–328.

[KS95] Y. Kornatzky and P. Shoval. Conceptual design of object-oriented schemes using the binary-relationship model. *Data & Knowledge Engineering*, 14(3):265–288, 1995.

[KW89] M. Kifer and J. Wu. A logic for object-oriented logic programming (Maier's O-logic revisited). In PODS89 [POD89], pages 389–393.

[Liu96] M. Liu. ROL: A deductive object base language. *Information Systems*, 21(5):431–457, 1996.

[Llo87] J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, Berlin, second, extended edition, 1987.

[LÖ90] Y. Lou and Z. M. Özsoyoglu. LLO: An object-oriented deductive language with methods and method inheritance. In J. Clifford and R. King, editors, *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data*, pages 198–207, 1990.

[LX93] K. J. Lieberherr and C. Xiao. Formal foundations for object-oriented data modeling. *IEEE Transaction on Knowledge and Data Engineering*, 5(3):462–478, June 1993.

[Mai86] D. Maier. A logic for objects. In *Workshop on Foundations of Deductive Databases and Logic Programming*, pages 6–26, Washington D.C., 1986.

[ME96] W. Y. Mok and D. W. Embley. Transforming conceptual models to object-oriented database designs: Practicalities, properties, and peculiarities. In Thalheim [Tha96], pages 309–324.

[ME98] W. Y. Mok and D. W. Embley. Using NNF to transform conceptual data models to object-oriented database designs. *Data & Knowledge Engineering*, 24(3):313–336, 1998.

[MGG95] R. Missaoui, J.-M. Gagnon, and R. Godin. Mapping an extended entity-relationship schema into a schema of complex objects. In Papazoglou [Pap95], pages 205–215.

[MIR93] R. J. Miller, Y. E. Ioannidis, and R. Ramakrishnan. The use of information capacity in schema integration and translation. In Agrawal [Agr93], pages 120–133.

[MIR94] R. J. Miller, Y. E. Ioannidis, and R. Ramakrishnan. Schema equivalence in heterogeneous systems: Bridging theory and practice. *Information Systems*, 19(1):3–31, 1994.

[MNE96] W. Y. Mok, Y.-K. Ng, and D. W. Embley. A normal form for precisely characterizing redundancy in nested relations. *ACM Transactions on Database Systems*, 21(1):77–106, 1996.

[MR89] H. Mannila and K.-J. Räihä. Practical algorithms for finding prime attributes and testing normal forms. In PODS89 [POD89], pages 128–133.

[MR92] H. Mannila and K.-J. Räihä. *The Design of Relational Databases.* Addison-Wesley, Wokingham, England, 1992.

[MR96] J. A. Makowsky and E. V. Ravve. Translation schemes and the fundamental problem of database design. In Thalheim [Tha96], pages 5–26.

[MR98] J. A. Makowsky and E. V. Ravve. Dependency preserving refinements and the fundamental problem of database design. *Data & Knowledge Engineering*, 24(3):277–312, 1998.

[NCB92] J. Nachouki, M. P. Chastang, and H. Briand. From entity-relationship diagram to an object-oriented database. In T. J. Teorey, editor, *Proceedings of the 10th International Conference on Entity-Relationship Approach*, pages 459–481. ER Institute, Pittsburgh, 1992.

[NNJ93] B. Narasimhan, S. B. Navathe, and S. Jayaraman. On mapping ER and relational models into OO schemas. In Elmasri et al. [EKT93], pages 403–413.

[NP88] S. B. Navathe and M. K. Pillalamarri. OOER: Toward making the E-R approach object-oriented. In Batini [Bat88], pages 185–206.

[ON95] Y.-C. Oh and S. B. Navathe. SEER: Security enhanced entity-relationship model for secure relational databases. In Papazoglou [Pap95], pages 170–180.

[Pap95] M. P. Papazoglou, editor. *Proceedings of the 14th International Conference on Object-Oriented and Entity Relationship Modelling*, Brisbane, Australia, 1995.

[POD89] *Proceedings of the Eighth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 1989.

[Prz88] T. C. Przymusinski. On the declarative semantics of deductive databases and logic programs. In J. Minker, editor, *Foundations of deductive databases and logic programming*. Morgan-Kaufmann, Los Altos, 1988.

[PTCL93] P. Poncelet, M. Teisseire, R. Cicchetti, and L. Lakhal. Towards a formal approach for object database design. In Agrawal [Agr93], pages 278–289.

[Qia96] X. Qian. Correct schema transformations. In P. Apers, M. Bouzeghoub, and G. Gardarin, editors, *Advances in Database Technology—EDBT '96*, number 1057 in Lecture Notes in Computer Science, pages 114–128. Springer-Verlag, 1996.

[Rei80] R. Reiter. Equality and domain closure in first-order databases. *Journal of the ACM*, 27(2):235–249, 1980.

[Rum87] J. Rumbaugh. Relations as semantic constructs in an object-oriented language. In N. Meyrowitz, editor, *Object-Oriented Programming Systems, Languages and Applications OOPSLA'87*, pages 462–481, Orlando, Florida, 1987. acm Press.

[SRSS93] D. Srivastava, R. Ramakrishnan, P. Seshadri, and S. Sudarshan. Coral++: Adding object-orientation to a logic database language. In Agrawal [Agr93], pages 158–170.

[ST93] K.-D. Schewe and B. Thalheim. Fundamental concepts of object oriented databases. *Acta Cybernetica*, 11(1-2):49–83, 1993.

[STH91] R. Spencer, T. J. Teorey, and E. Hevia. ER standards proposal. In H. Kangassalo, editor, *Proceedings of the 9th International Conference on Entity-Relationship Approach*, pages 425–432. Elsevier Science Publishers B.V (North-Holland), Lausanne, Switzerland, October 1991.

[Tha93a] B. Thalheim. Foundations of entity-relationship modeling. *Annals of Mathematics and Artificial Intelligence*, 1993(7):197–256, 1993.

[Tha93b] B. Thalheim. Towards a theory of cardinality constraints. Rostocker Informatik-Berichte 14, Universität Rostock, 1993.

[Tha96] B. Thalheim, editor. *Proceedings of the 15th International Conference on Conceptual Modeling*, number 1157 in Lecture Notes in Computer Science, Cottbus, Germany, 1996.

[TYF86] T. J. Teorey, D. Yang, and J. P. Fry. A logical design methodology for relational databases using the extended entity-relationship model. *ACM Computing Surveys*, 18(2):197–222, 1986.

[Ull88] J. D. Ullman. *Principles of Database and Knowledge-Base Systems (Volume I).* Computer Science Press, Rockville, MD, 1988.

[vBW94] M. F. van Bommel and G. E. Weddell. Reasoning about equations and functional dependencies on complex objects. *IEEE Transaction on Knowledge and Data Engineering*, 6(3):726–768, 1994.

[VdB93] J. Van den Bussche. *Formal Aspects of Object Identity in Database Manipulation.* PhD thesis, Universiteit Antwerpen, 1993.

[VS93] M. W. Vincent and B. Srinivasan. Redundancy and the justification for fourth normal form in relational databases. *International Journal of Foundations of Computer Science*, 4:355–365, 1993.

[Wed89] G. E. Weddell. A theory of functional dependencies for object-oriented data models. In Kim et al. [KNN89], pages 165–184.

[Wed92] G. E. Weddell. Reasoning about functional dependencies generalized for semantic data models. *ACM Transactions on Database Systems*, 17(1):32–64, March 1992.