Chapter 3

Concepts and Relations

Abstract

The Business Rules Manifesto proclaims as its central thesis or "mantra": Rules are built on facts, and facts are built on terms. But what are they, the terms, facts and rules that the Manifesto talks about? How must you understand them, how are they interconnected, and why can you rely on just these notions to build a rigorous framework for business rules? The answer is found in mathematics: Relation Algebra is a formal method to define, implement and work with well-formed business rules. Relation algebra provides todays professionals with a way of thinking and a way of working: what to do to describe and manipulate relations effectively, quickly, and without mistakes. Proficiency in these skills will allow you to analyze business requirements, to conduct conceptual analysis, to search for and create new rules, to test the outcome, and finally to produce exact specifications.

This chapter explains almost all about two basic notions that we will be needing as we identify, express, and manipulate the rules in the business environment. These two notions are: concept and relation.

We will introduce them using a language-oriented approach. Which is natural, as business workers express their requirements and ways of working in natural language. But to turn that into very exact rules that can be handled by computers, we cannot escape using a mathematical formalism, called Relation Algebra. Relation Algebras as a distinct branch in mathematics came about in the nineteenth century by the efforts of mathematicians such as De Morgan, Peirce, and Schröder [9, 27, 30]. Their results have been studied, expanded and enhanced throughout the twentieth century. This has resulted in a clear and well-understood formalism that meets our needs in describing business rules.

After reading this chapter, you are expected to be familiar with these notions because all business rules approaches, and the Ampersand approach is no exception, are built upon these fundaments. There are many excellent articles and introductions for those who want to study the mathematical body of Relation Algebra. The 'background' material of Jipsen, Brink, and Schmidt is recommended as a reference for relation algebras in general. We will specifically use the so-called heterogeneous relation algebras. Roger Maddux's book is an excellent resource for a further study of relation algebras.

This chapter is outlined as follows. We start from fundamentals: simple sentences. Next, we discuss some mathematical notions that the Business Rules Manifesto implicitly uses in its article 3.1 "Rules build on facts, and facts build on concepts expressed by terms". Examples will illustrate the theory, in order to give a better understanding of the ideas and formalisms. And do not worry, we only need a basic and almost intuitive understanding, no advanced mathematics is involved.

3.1 Sentences

3.1.1 Simple sentences

Business workers use natural language to talk about things in the real world. Therefore, we need a clear and unambiguous way to capture the meaning in natural language sentences. Let us consider the following simple sentences to start with:

– ABBA sang the song Waterloo,

– DoeMaar sang Smoorverliefd,

– Joosten receives building-permit number 5678,

– William Kennedy has residence-permit NL44,

– ABBA sang 'Money, money, money',

– Ersin Seyhan has residence-permit NL901,

– the Open University offers the course Business Rules,

– Fatima has passed the exam for Business Rules,

– Caroline has passed the exam for Spanish Medieval Literature.

These sentences follow the scheme: noun - verb - noun, in which the nouns refer to objects in the real world. The Business Rules Manifesto uses the word *term* to refer to the individual real-world objects such as *term*

ABBA, the building permit number 5678, the residence permit NL_44, Caroline, and the Business Rules course.

Instead of 'term', many other words are in use: software engineers call them 'instance' and mathematicians call them 'element'. In knowledge engineering and model theory, such things are called 'atom'. Throughout this book we use the word 'atom', though any synonym like 'instance', 'item', 'element', 'member' or 'object', will do in practice.

atom

An *atom* refers to an individual object in the real world, such as the student called 'Caroline'. But what if there are three different Carolines? What does it mean to say: "Caroline has passed the exam for Spanish Medieval Literature."? This sentence might be true for one Caroline, but false for the others. Clearly, to avoid ambiguous sentences, an atom must identify exactly one real-world object, no more, no less. Or rather, it suffices that the atom identifies one object within the context in which we are working: if the context is a group with only one Caroline, there will be no ambiguity. Similarly, ABBA is unique among all pop groups in the world; there ought to be only one building permit with number 5678; etcetera.

fact

Each of the simple sentences above represents a *fact*, something that is taken to be true. The example sentences follow the noun-verb-noun scheme, but the notion of simple sentence applies to just about any sentence that has two atoms related by a verb-like expression. For example, if we use the name 'Caroline' and the number '3' as atoms, the following is also a simple sentence: "In Caroline's house there are three rooms". This simple sentence contains two atoms, and if we leave out those atoms, a template remains, as in: "In ... 's house, there are ... rooms". So, a *simple sentence* in natural language relates two atoms.

simple sentence

Still, a simple sentence does not accommodate arbitrary atoms. For example, switching the atoms produces a sentence "In 3's house, there are Caroline rooms" which makes no sense at all, it is non-sense. In this particular example, the first blanks are clearly meant for a person and the second one should obviously be a number. In other sentences, other concepts may be required. For instance, in the sentence "... has passed the exam for ..." the first blanks are supposedly some student and the second blanks must be some course.

concept

In Ampersand, an abstraction like 'student', 'number', and 'course', that you need to replace by an actual student, number, or course, is called a *concept*. Concepts should not be confused with atoms: concepts are abstract, and atoms are concrete. Atoms are the terms, the individual things in the real world that you can point out. A concept is an abstraction, it is not the individual thing but the type of thing. We can say for example that Caroline (an atom) is a student (a concept), ABBA is a pop group, and NL_44 is a residence permit.

So simple sentences have a fixed template, and moreover, every term (or atom, as we prefer to call it) that you fill in on the blanks, belongs to a certain concept. The Business Rules Manifesto, in its article 3.1, hints at just this distinction between term and concept. That article can be explained as:

– a *term* expresses a business concept, it corresponds to an individual thing that is relevant in the business context,

– a *concept* is an abstract description or definition that is instantiated (expressed) by the actual realworld thing (term or object),

– a *fact* makes an assertion about the concepts, it is a simple sentence that expresses a relationship between two terms.

3.1.2 Concept

As we want to describe our business rules with computer precision, we must be rigorous in our definitions.

At the surface level, a *term* refers to one individual thing that exists in the real world. At the deep level, a *concept* stands for terms that are similar: they share the same meaning, have similar properties, similar behavior, and similar connections with other terms or concepts.

intension

Every abstract concept comes with an *intension*: a definition that lets you determine whether an arbitrary 'thing' in the business environment is an instance of the concept. The reader of the definition is expected to understand which 'individual terms that exist in the real world' meet the definition and should be (represented as) a term in the extension of the concept. Next, a concept also has an *extension*, more often called *population* or *contents*. This is the set of all terms that the concept actually stands for, at present. It is customarily denoted using a *bracket notation*, for instance $Age = \{ 7, 3, 5, 2 \}$. Occasionally square brackets are used: $Age = [5; 3; 2; 7]$. In mathematics, the special symbol $\in$ is used to denote that an element is contained in the set, so we have: $3 \in Age$, $7 \in Age$, etc.

extension
population
contents

inclusion

The *inclusion* symbol $\subset$ is used to denote that one set is a subset of another, it is contained in it. For instance $\{ 3, 7 \} \subset Age$, or { William Kennedy, Ersin Seyhan, Caroline } $\subset$ Person. Because the inclusion symbol is not easy to type, we will often replace it with the symbol $\vdash$, which means exactly the same.

Subsets usually contain less atoms than the enveloping set, but in general, the two sets are allowed to be equal. If we want to draw attention to the fact that equality is permitted, we sometimes write $A \sqsubseteq B$. But if equality is not permitted, we must write two assertions:

$$A \subset B \text{ and } \neg (A = B)$$

Three important properties to remember about atoms are:

– each instance of the concept is considered to be whole and indivisible, and it is why we use the name 'atom'. The element 'ABBA' is regarded as one indivisible member of the Popgroup concept, even though, from another point of view, it may be composed of four units,

– there is no specific order or sequence among the atoms of a concept,

– an atom can occur only once in the population. That is: each atom must differ from every other element within the set. A term meets the concept definition, or it does not. It is meaningless to say that it meets the definition twice.

entity integrity

The latter property is sometimes referred to as *entity integrity* by database designers.

A concept is characterized by its intension, a sound definition that exemplifies its business semantics, its meaning, properties, behaviour. Definitions are generally stable, and do not change overnight, regardless whether there are millions, hundreds, tens or even no instances on record for the concept. And although definitions can change over time, such changes are not very frequent.

empty extension

To contrast: extensions can and will change constantly. The extension is the time-varying part. At any given moment, there is a specific content, and that content can differ from the content one minute before or after. For instance, Student is an important concept at any university, but the student population is constantly changing. A concept may even have an *empty extension* at some time, denoted $\emptyset$. But even if two concepts are both empty, we consider them to be different: a Student concept differs from the Car concept, even if there are neither students nor cars on record.

In general, concept names are nouns such as 'Student', 'Customer' or 'Car', sometimes qualified to better express its meaning, e.g. 'Old-timer Car', 'Bachelor Student', or 'Priority Customer'. By convention, we write the name of a concept with a capital: 'Popgroup', 'Permit', or 'Course'.

3.1.3 Relation

Above, we defined 'fact' as a simple sentence that expresses a relationship between two terms. Having abstracted the individual terms to concepts, we also desire an abstract notion to replace the individual fact templates. The answer provided by mathematics is called: relations.

Let us introduce this by way of an example: some friends organizing a tennis tournament which will include a mixed-double competition. Although the group of friends has 4 women and 5 men, only three pairs want to participate in the mixed double. These are the facts:

- Aisha doubles with Marek.
- Sophie doubles with Raúl.
- Nellie doubles with Toine.

Obviously, the template for these facts is: ... *doubles with* ... with the first concept being Woman, the second concept involved is Man. We now introduce the relation named *doublesWith* which is defined as "the set of all pairs of one woman and one man wanting to participate in the upcoming tennis tournament". We can write the pairs one by one:

- ⟨`'Aisha'`,`'Marek'`⟩ $\in$ *doublesWith.*
- ⟨`'Nellie'`,`'Toine'`⟩ $\in$ *doublesWith.*
- ⟨`'Sophie'`,`'Raúl'`⟩ $\in$ *doublesWith.*

Or we may list the entire population of *doublesWith* in one go:

$$\mathit{doublesWith} \;=\; \{\ (\text{Nellie}, \text{Toine});\ (\text{Aisha}, \text{Marek});\ (\text{Sophie}, \text{Raúl})\ \}.$$

We can also make a drawing of this relation, a so-called *instance diagram* or Venn diagram. This shows all facts as arcs connecting the woman and man that together make up a pair, as shown in figure 3.1. An instance diagram provides very detailed insight into the current state of affairs in the business, which can be very useful sometimes. It works fine for small examples, but this kind of diagram quickly becomes unreadable if many pairs participate.

instance diagram

We may also use a layout with two columns as in table 3.1. Facts can be represented in many ways. A telephone directory lists names with telephone numbers in a tabular layout, a dictionary does the same with words and their meanings. But a story about a poker or bridge game may be illustrated with a diagram showing the hands of each player.

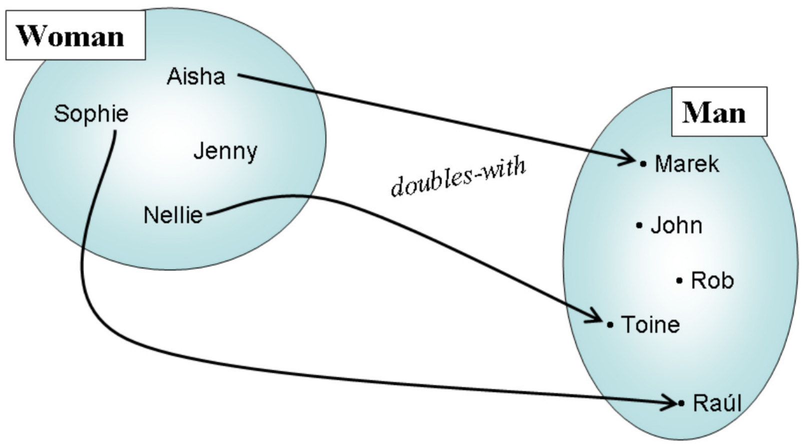

Figure 3.1: Instance diagram: which woman *doublesWith* which man

Woman	*Man*
Sophie	Raúl
Aisha	Marek
Nellie	Toine

Table 3.1: *doublesWith*: tabular layout of the participating pairs

The time and effort people take to communicate the contents of relations indicates the importance of suitable representations. In practice, form follows function: depending on circumstances, audience, and the sheer number of facts, the designer chooses a representation that best suits the purpose.

3.1.4 Cartesian product

The most comprehensive representation of a relation is a two-dimensional array. It shows all instances of one concept, Woman, along one axis, and all instances of the second concept, Man, along the other axis, like a spreadsheet page as in table 3.2. It allows you to easily mark participation in the tournament.

This representation lets you consider all possible combinations of one woman and one man, a total of $4 \times 5 = 20$ possible pairs. The "set of all possible pairs" is called a *Cartesian product* in mathematics.

Cartesian product

Formally, the *Cartesian product* of two concepts A and B is (the set of) all pairs that combine one atom from the first concept A, with one atom from the second concept, B. Obviously, the number of pairs in the

doublesWith	Marek	John	Rob	Toine	Raúl
Sophie	-	-	-	-	x
Aisha	x	-	-	-	-
Jenny	-	-	-	-	-
Nellie	-	-	-	x	-

Table 3.2: Two-dimensional layout of the participating pairs

Cartesian product is equal to the number of atoms in A times the number of atoms in B; which explains (in part) why it is called a 'product'.

The Cartesian product of concepts A and B is denoted as: $A \times B$. We sometimes denote it as $\mathbb{V}_{[A,B]}$ or $\mathbb{V}$ for short, with the V's lefthand side rendered as a double line, if the printer can do that. Of course, if there is any chance of ambiguity, $\mathbb{V}_{[A,B]}$ should be written in full to prevent confusion with, say, $\mathbb{V}_{[B,C]}$ or $\mathbb{V}_{[B,A]}$.

Another good example of Cartesian product is a deck of common playing cards. There are 4 suits {spades, diamonds, clubs, hearts}, and 13 ranks. Because every combination is possible, there are 52 playing cards in the Cartesian product of *Suit* $\times$ *Rank*, as shown in figure 3.2. The cards in the picture are neatly arranged, but remember that in general, there is no special ordering along the axes.

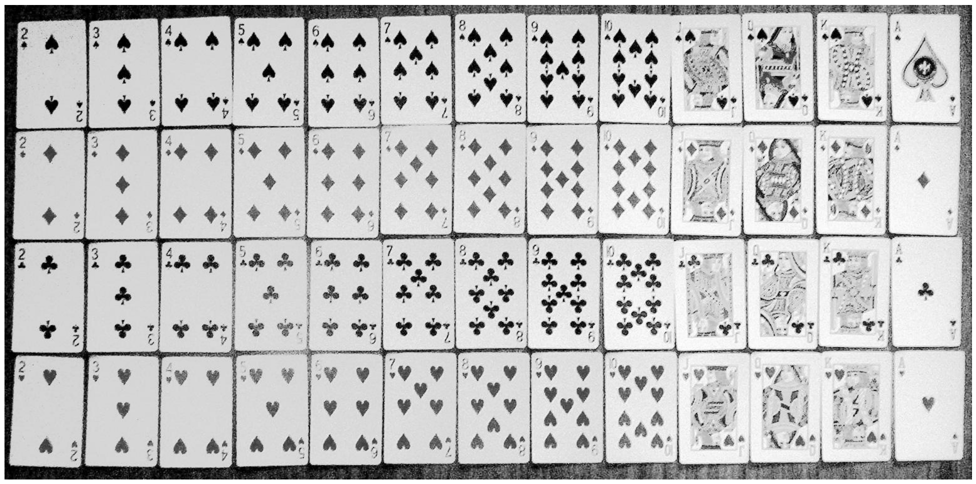

Figure 3.2: Set of 52 playing cards

To be exact: this is different from *Rank* $\times$ *Suit*. While the tuple (diamonds, ace) looks very similar to the tuple (ace, diamonds), there is a big difference as the two atoms are written in inverse order. This finesse will be important later on when we discuss the inverse of a relation.

Some standard terminology when discussing Cartesian products:

source

– the left-hand set of the product is called *source* or *domain*,

target

– the right-hand set of the product is called *target*, or *co-domain*, or sometimes *range* or *image*,

tuple

– each single element of the product is called a *tuple* or *pair*.

3.1.5 Relation in Mathematics

relation

Having defined the Cartesian product, we can now explain the formal definition of a *relation* which is:

– a named subset of the Cartesian product of two concepts, where all tuples in the subset have a similar meaning, similar properties, and similar behavior.

relation name

The *relation name* is usually a verb, possibly qualified, like you saw in the templates for simple sentences. We will generally write the relation name in italics and in lowercase. Whatever name you choose for a relation, make it (almost) self-explanatory. When trying to understand a design, the business stakeholders will start to look at names first. So make sure that the names are really close to the intent of the relation, its real-world meaning and pragmatics.

intension

pragmatics

Like with concepts, a relation has an *intension*, a precise definition. The definition serves to determine which pairs should be represented in the extension of the relation, and which ought to be absent. The definition or *pragmatics* captures the meaning of the relation.

You should check that *doublesWith*, with its three couples entered for the tennis competition, is indeed a relation according to our definition. Another example of a relation is the subset of community cards lying face-up on the table in a poker game, shown in figure 3.3.

A tuple in a relation refers to one instance of the source and one instance of the target, and the tuple is fully identified by exactly these two atoms. Therefore, there is no need to have special identifiers to distinguish between the tuples in a relation: the identification of source- and target instance suffices.

To summarize so far:

– an *atom* corresponds to an individual real world thing. Each atom or *term* is captured as an instance, an element in a set, and that set is the extension of a concept,

Figure 3.3: Community cards on the table are a special selection of playing cards

- a *concept* is an abstract description or definition that is instantiated by the actual realworld things or objects. The intension is the definition of the relevant terms; the extension is the set of all terms in the business context that meet the definition,
- a *fact* is a simple sentence that expresses a truth about two terms,
- a *relation* is an abstract description of facts. Mathematically, it is defined as a subset of the Cartesian product of two concepts, and it captures the deep structure of simple sentences,
- the *intension* of the relation provides the definition or meaning of the relevant facts,
- the *extension* of the relation consists of *pairs*, and each *tuple* represents a single fact that is true in the business context.

3.1.6 Terminology and notations

Some standard terminology when discussing relations:

- the relation *declaration* is the combination of its name, its source and its target, together with its definition, i.e. its meaning or intension, *declaration*
- the *signature* of a relation is the combination of relation name, (the name of) source concept, and (the name of) target concept, *signature*

type

– the *type* of a relation is defined as (the name of) the source concept, and (the name of) the target concept. In other words: a relation type indicates the full Cartesian product, the relation is its subset.

Type is important when we compare the extensions of relations. If two relations have different types, then they contain tuples with atoms taken from different source or target concepts. Only if two relations are of the same type, i.e. both relations are subsets of the same Cartesian product, can we compare their contents.

To avoid ambiguity in discussions, every relation should have a unique signature, i.e. the name, source and target are a unique combination. Often, if sources and targets are not in doubt, the relation name is enough to know about which relation we are talking. Sometimes one name is used for several relations. Theoretically, this is fine provided that the types are different. For stakeholders, it may be rather confusing.

To denote the signature of a relation with name r, source A and target B, we write

$$r : A{\times}B \text{ , or } r_{[A,B]} \text{ for short.}$$

The customary notation for writing the contents of a relation is

$$r = \{ \ (\ a,b \ ) \mid (\ a,b \ ) \in \ r\}$$

A more concise notation is sometimes used for a single tuple:

$$a \ r \ b \text{ means that tuple } (\ a,b \ ) \in r$$

For a given $a \in A$, we can determine all elements $b \in B$ that are related to a. This set, which in general may have 0, 1 or any arbitrary number of elements, is called the target of the element a:

$$\mathit{target}(a) = \{ \ b \in B \mid (\ a,b \ ) \in r\}$$

Likewise, for a given $b \in B$, the subset of instances in A that relate to b is called the source of b:

$$\mathit{source}(b) = \{ \ a \in A \mid (\ a,b \ ) \in r\}$$

Notice in the above formulas that the left-hand sides do not indicate about which relation we are talking. To avoid confusion, we may indicate the relation for which the target and source are being considered:

$$target_{[rela,tion]}(a), \text{ and } source_{[rela,tion]}(b)$$

So now the words ‘source’ and ‘target’ have double usage. First, every relation r has a source and a target concept. And second, each atom of the source has its own target set, and each instance of the target has its source set.

Bear in mind that in general, a relation is not the entire set of all possible combinations of elements, but only a subset, a selection of certain pairs. To emphasize this important difference, the Cartesian product $\mathbb{V}_{[A,B]}$ is sometimes called the *universal relation* of A and B, with "universal" meaning that it contains all possible tuples that can be produced from (the current contents of) A and B.

universal relation

Various mathematical *symbol*s will be needed further on. The symbol $\forall$ means 'for all', $\exists$ means 'there exists', $\wedge$ means 'and', $\vee$ means 'or', $\rightarrow$ means 'implies', and $\leftarrow$ means 'is implied by', which is the same but it goes in the other direction. These symbols are merely a mathematical shorthand notation. Just remember the meanings and pronunciations above, and you will find nothing mysterious about them.

symbol

3.1.7 Identity relation

At this point, we want to introduce to you a special relation: the so-called *identity relation*, abbreviated to Id or even to $\mathbb{I}$ (for this particular relation yes, an uppercase i).

identity relation

This relation can be defined for every concept, taking that concept for its source as well as for its target. The definition is very simple: it states that each atom (of the source) is equal to itself (now as an atom in the target). The contents of this relation is equally simple: it contains every possible tuple composed of two identical elements.

identity	Apple	Orange	Pear	Berry	Grape
Apple	x				
Orange		x			
Pear			x		
Berry				x	
Grape					x

Table 3.3: Identity relation is special selection of the Cartesian product with identical source and target

In the matrix representation of table 3.3, the identity relation has a marking exactly on the diagonal, and nowhere else. The example tells us five facts: 'fruit w is identical to fruit w', for all five fruits, ranging from apple to grape. Another way to write down this identity relation is as follows:

$$\mathbb{I}_{Fruit} \;=\; \{(\; w, w \;) \mid w \in \mathit{Fruit}\;\}$$

The subscript indicates the concept for which the Identity relation is taken. If the concept is not in doubt, the subscript is usually omitted.

3.1.8 Naming

We aim to use the notions of atom and fact, concept and relation consistently throughout this book. Other authors coin other names and notions that may look rather similar in meaning and usage, but often there are subtle differences.

For instance, UML programming works with ‘classes’, a notion that is similar but not equal to our notion of concept. And database modelling uses the name ‘entity’, or sometimes even ‘type’, for a notion that is also similar but not equal to our concept.

The SBVR standard (more on that later) defines term as a ‘verbal designation of a general concept in a specific subject field’. The ‘general concept’ that this alludes to, is described as a ‘unit of knowledge created by a unique combination of characteristics’.

The many different names and definitions may be a cause of confusion. Illustrative of this is the SBVR statement that “a concept type is an object type that specializes the concept ‘concept’, whereas a concept is related to a concept type by being an instance of the concept type.”

So according to the SBVR naming convention, a person John Doe should be called a concept, which is an instance of the concept type *Customer*. We however prefer to call *Customer* the concept, and refer to the person John Doe as an instance of Customer.

Also, what we call a relation goes under different names. SBVR calls ‘fact type’ for what we prefer to call a ‘relation’. UML coins the word ‘association’, and ‘reference’ and ‘relationship’ are used by still others.

For some readers, words like ‘source’ and ‘target’ may bring to mind the hyperlinks of the World Wide Web. However, hyperlinks, strictly speaking, are not relations. Although a hyperlink does provide a link from a source (webpage) to a target (another webpage), the reverse, from target back to source, is missing. For a given webpage, there is no sure way to know all webpages that have a hyperlink to it. Moreover, hyperlinks do not implement the referential integrity demand (see below), resulting in the infamous “error 404” reports.

3.1.9 Integrity

When you deal with concepts and relations in computers, you must always adhere to two demands.

First, for every concept, it must be so that *every atom is unique within the extension.* Remember that each atom represents a single term in the business environment. That term meets the definition of the concept, or it does not. It is a member in the population, or it is not. But never can it be in the population, twice! If students are identified by their first names, then the computer can only deal with one student named Caroline. This demand was referred to as *entity integrity*.

entity integrity

Second, we stated that each tuple in (the extension of) a relation refers to one instance of the source and one instance of the target concepts. This requires that those two *atoms referred to must actually exist* in the current extensions of the concepts, they must be on record. This important demand is usually referred to as *referential integrity*.

referential integrity

At first glance, these two are rather trivial demands. However, one must realize that extensions are constantly changing. Therefore, if an atom is deleted from (the current extension of) some concept, the consequence is that all tuples referring to that very atom ought to be deleted also, in all relations that the concept may be involved in, either as a source or as a target. On the other hand, whenever we want to insert a tuple in a relation, we must assure that the associated source and target atoms are present in their respective extensions.

3.1.10 Validation

The intension (definition) of a relation tells us how to decide which facts ought to be true in the business environment. The extension is the current content, it captures all of the facts that are currently on record.

You must realize that these two are fundamentally different. For example, the current content of the relation may not be up-to-date: a fact is true, but not yet recorded. Or a tuple is recorded for a relation, but the corresponding fact is no longer true. This kinds of problems may not always be attributed to a computer issue or a business error, e.g. a customer has posted a complaint but the mail has not yet been received.

The activity to check whether the computer-stored information is a valid representation of the situation in the real business world is called *validation*. Such a check will concern the (current) contents of one or more extensions. Validation may also be used in a broader sense: whether an entire design captures all the relevant features of a business environment. Validation in general requires a human effort, as only humans have a grasp of reality. Only on rare occasions can the computer signal validation problems.

validation

Do not confuse validation with *verification* which is only an internal check on the stored information, and reporting possible problems, especially problems with integrity. Such checking can be done automatically, by computers, without referring to the real world. Clearly, problems detected by verification may sometimes be caused by invalid data, but there may also be errors in the data store or in the programmed checks.

verification

3.2 Models and diagrams

3.2.1 Conceptual Model

If you are trying to understand a business context, you will need to get a grip on the concepts and relations that are important. You start out with simple sentences, try to extract the deep structure of the sentences, and keep note of the structures you have uncovered so far.

conceptual model

A *conceptual model* is the exhaustive listing of all concepts and relations that are relevant in a certain (business) setting. Such a listing can be provided in textual form, which will make it dull and incomprehensible, and only a trained engineer or computer programmer will like it. The listing can also be provided in a more attractive way, such as a diagram, a graphical representation of the model, or even by way of a prototype system for users to explore and play with.

For a conceptual model to be correct, we require that the signature of each relation shall be unique: for any given source and target, there is at most one relation with a certain name. The same name may be used elsewhere in the model, for relations defined on other sources or targets. However, it can become quite confusing for people trying to read and understand a model if different relations have identical names.

3.2.2 Conceptual diagram

A good way to get a grip on the concepts and relations in your business environment is by making a diagram such as figure 3.4. A picture is often very useful to oversee the context, and to discuss important aspects with stakeholders.

The example conceptual diagram uses dots to represent concepts, connected by arcs that represent the relations. This diagram employs the arrow notation, with the arrowhead pointing from the source (shaft of the arrow) to the target concept (point of the arrow). Relation names are written next to each arc, so that the unique signature of each relation (name, source and target) is easy to read from the diagram.

Remark that the diagram does not show current students or courses that are presently available. Nor does it show which students are involved in which course. In general, abstracting away from the contents enables you to oversee the structure of many relations at once.

The diagrams are easy to understand as long as the number of concepts and relations is moderate. When the numbers go up, say more than 20

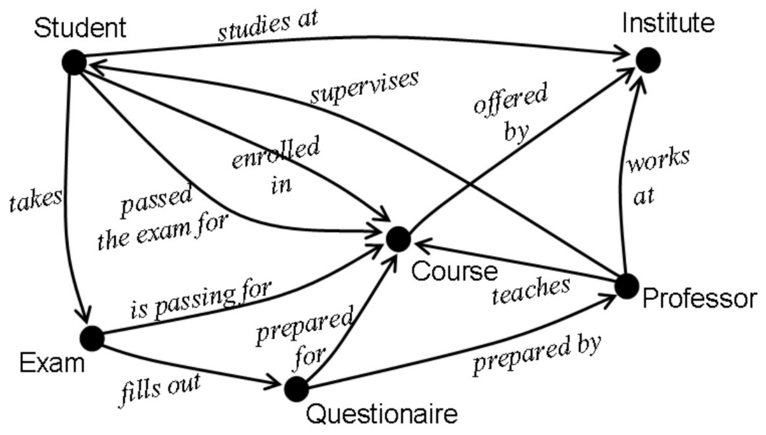

Figure 3.4: Example of a conceptual diagram

concepts, then the sheer size of the diagrams makes them incomprehensible for most people, and again, only the trained specialists will like to work with such diagrams.

In this kind of diagrams, the arrowheads are sometimes placed at the base of the arc (as in the 'crow's feet' notation well known from database modelling). Other notations place it halfway, or at the end of the arc, as for instance is common in UML diagrams. Still other notations omit the arrows altogether, so that source or target must be looked up in the documentation of the conceptual model.

3.2.3 Instance diagram

You can also use diagrams to show (some of) the contents of the concepts and relations. Such a diagram is called an Instance diagram, as shown in figure 3.5. The instance diagram provides a level of detail that is lacking in the Conceptual Diagram. It depicts some instances of the concepts and relations, corresponding to true facts of reality. Already at this small scale, the diagram is rather cluttered. As mentioned before, instance diagrams are generally not very comprehensible due to the sheer amount of detail. They are mostly used only with small subsets, to illustrate a certain finesse of argument or to highlight some intricacy.

3.3 Operations on relations

We started out with simple sentences, facts that are true within a certain business context. Sentences can be combined in many ways to produce

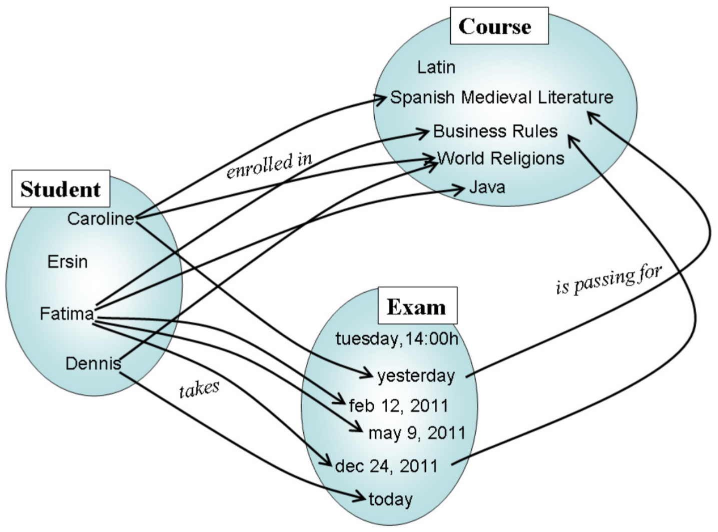

Figure 3.5: Example of an instance diagram

other sentences, that may be less simple to understand. But if you make proper combinations, then the new sentences will express something about the business that is true as well.

Relation Algebra provides numerous operations to produce new relations from existing ones. Learning to use them is vital in order to exploit the expressive power of relations: applying the operators correctly will ensure that the new sentences, your new facts, will be true as well.

We begin with two simple yet important operators: complement and inverse. Then we discuss operations that you ought to be familiar with: intersection, union, and difference. The most important operator is discussed next: composition. Other relational operators such as 'dagger' and 'implication' are discussed thereafter.

3.3.1 Complement

complement operator

The *complement operator* is used to indicate the atoms that are *not* in the population of a relation.

Consider the mixed doubles, defined as subset of the Cartesian product of Woman $\times$ Man. The relation signature is $doublesWith_{[Woman,Man]}$. The complement is the relation that has the same type, but its contents is the 'remainder': all pairs in $\mathbb{V}$ that are *not* in the original relation. For this reason, some authors refer to this operation as *negation*.

negation

Complement is usually denoted by an overstrike, $\overline{r}$, and you pronounce it as ‘complement of r’. Because overstrike is not easy to type, it may be denoted by a minus sign in front of the relation: -r.

A tabular representation of the complement relation is easy to write down. In the example of teams for the mixed double, it contains a total of $4 \times 5 - 3 = 17$ tuples, representing all potential couples that are not entered for the mixed doubles competition (table 3.4).

complement of *doublesWith*	Marek	John	Rob	Toine	Raúl
Sophie	x	x	x	x	
Aisha		x	x	x	x
Jenny	x	x	x	x	x
Nellie	x	x	x		x

Table 3.4: Pairs not entered for a mixed double competition

set difference

Another way to denote a complement is by way of *set difference*, using the backslash \symbol. The set difference $F\backslash G$ is all instances that are contained in set F, at the left, but not contained in the righthand set, G. So we may denote the complement of relation $r_{[A,B]}$ as the difference $\mathbb{V}_{[A\times B]}\backslash r$.

Obviously, if there is any ambiguity what the enveloping Cartesian product would be, then the complement would also be in doubt. To illustrate the point: the complement of all students attending class is probably those students that are enrolled for the course, but that do not attend todays class. But perhaps the speaker meant the people in the classroom that are not students, such as the teacher and perhaps a visitor?

involutive

Complement is a so-called *involutive* operator: apply the operator twice, and it will reproduce the original relation. In formula:

$$\overline{\overline{r}} = r$$

3.3.2 Inverse

Earlier, we pointed out that the Cartesian product of a set A and a set B is $A \times B$, and this is not the same as $B \times A$ (except of course if $A = B$, more on that later). However, a relation that contains tuples of the form (a, b) can easily be altered into a relation containing tuples of the form (b, a) simply by changing the order of the elements. Mathematically, it is a brand new tuple, as source and target differ from before.

inverse
conversion

This operation, producing a new relation from an existing one, is called ‘taking the *inverse*’, inversion, or sometimes *conversion*. It is denoted by writing a $\smile$ symbol (pronounced ‘flip’) after the relation name:

$$sang^{\smile}_{[Hit\ song, Popgroup]}$$

We may pronounce this as ‘flip of Popgroup *sang* Hit song’. For most of us, it is more comfortable to change the relation name into something more sensible like ‘Hit song *wasSungBy* Popgroup’, or ‘Hit song *originatedFrom* Popgroup’. Beware however that in general, different relation names mean different relations!

The formal declaration of the inverse relation $r^{\smile}$ of a relation declared as $r_{[A,B]}$ is as follows:

$$r^{\smile}{}_{[B,A]} \text{ having contents} \{ (\ b, a\ ) \mid (\ a, b\ ) \in r\ \}$$

As the inverse operator merely swaps the left and right hand sides of each tuple, it is easy to write down the extension of the new relation, if given the extension of the old one. In a tabular form, the inverse operator merely swaps columns, as shown in table 3.5.

Man	*Woman*
Raúl	Sophie
Marek	Aisha
Toine	Nellie

Table 3.5: Inverse of the selected couples for a mixed double competition

In the two-dimensional matrix layout (see table 3.2), the inverse operator mirrors the entire content of the matrix along the diagonal. The two axes, source and target, are swapped accordingly.

involutive

Like complement, the inverse operator is also *involutive*: apply it twice, and you end up with the original relation. This may be no surprise, as the inverse operator does not change the facts, it merely rephrases them. In formula:

$$r^{\smile\smile} = r$$

3.3.3 Set operations

Set theory in mathematics describes several operators that produce new sets from existing ones. Each operation unambiguously defines which elements will belong to the new set, and which ones are excluded.

In particular, we assume that you are familiar with the intersection, union and difference operators on sets. It ought to be an easy exercise to verify for arbitrary sets F and G that:

$$F \cup G = (F \backslash G) \cup (F \cap G) \cup (G \backslash F)$$

and

$$(F \setminus G) \cap (F \cap G) = \emptyset$$

Because we defined relations as subsets of a Cartesian product, we can apply any *set operation* to two relations r and s, provided of course that they share the same types. We have:

set operation

intersection

- *intersection* : $r \cap s$ is the relation that contains the tuples that are contained in relation r as well as in s, or $r \cap s = \{(x,y) | (x,y) \in r \text{ and } (x,y) \in s\}$

union

- *union* : $r \cup s$ is the relation that contains all pairs that are present either in relation r or in s, or $r \cup s = \{(x,y) | (x,y) \in r \text{ or } (x,y) \in s\}$

difference

- *difference* : $r \backslash s$ is the relation that only takes tuples of relation r not present in s, or $r \backslash s = \{(x,y) | (x,y) \in r \text{ and } \neg (x,y) \in s\}$. You may check that this is equivalent to the intersection $r \cap \overline{s}$.

Remember, if relations are not defined on the same Cartesian product, they will contain tuples that cannot be compared. Such relations are disjoint, and the set operations above produce no meaningful results.

3.3.4 Composition

The *composition* operator is perhaps the most important operation in Relation Algebra. It produces a new relation from two existing ones. Or more down to earth: it combines two facts, two simple sentences into a single sentence that is not so simple any more. The formal definition of composition is as follows.

composition

Let $r_{[A,B]}$ and $s_{[B,C]}$ be two relations, with the same concept being the target of r as well as the source of s. Then the composition of r and s, is the relation with type A $\times$ C. Its content is calculated as

$$\{ (a, c) \mid \text{there exists at least one } b \in B \text{ such that } a\ r\ b \text{ and } b\ s\ c\}$$

The composition operator is denoted by a semicolon ; between the two relation names, like this: $r \ ; \ s$. It is pronounced as 'composed with', in this case: r composed with s.

The figure 3.6 illustrates how composition works. An Actor (at the left) *possesses* some Skills (middle), and Skill *required-for* a Role (at the right). The meaning of the composition relation is: the Actor *possesses at least one of the Skills required for* the Role. In natural language, you might say that the actor has some skill for the role, but the formalization is much more precise. It is a combination of two facts: the actor possesses

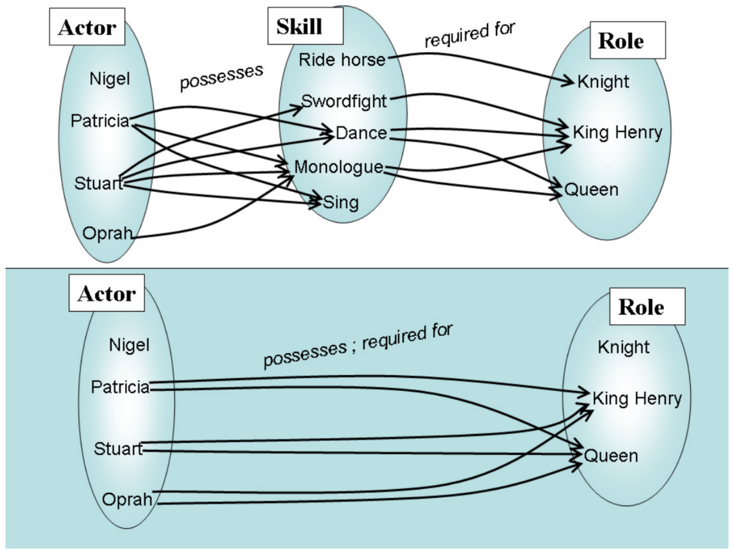

Figure 3.6: Instance diagram of a composition

a particular skill. And the skill is required for the role. In fact, the actor may possess several skills, or the role may require many skills, but exact information about skills is absent from the composition. For an actor to be related to a role means only that the actor has at least one required skill for the role.

Another example was seen in figure 3.5 by composing relation *takes* with *isPassingFor*. Student Caroline takes an exam, yesterday, that is passing for Spanish Medieval Literature. Also, student Fatima takes several exams, and one of them is passing for the Business Rules course. But according to the diagram, no exam was taken by Dennis that is passing for World Religions.

Obviously, you cannot compose just any two relations. If there is no middle ground, then composition has no meaning. Readers familiar with relational database theory will recognize that composition corresponds to the 'natural join' operator.

3.3.5 Advanced operators

Apart from composition, several other operations can be defined for two relations. We will briefly discuss them, although we will rarely use them. Moreover, mathematically speaking, it can be shown that all these advanced operations can be reduced to ordinary composition. In the following definitions, let $r_{[A,B]}$ and $s_{[B,C]}$ be two relations, with B being the target of r and the source of s.

The *relative addition* of r and s, is the relation with signature $(r \dagger s)$: $A \times C$, and its content is defined as *relative addition*

$$\{ \ (\ a, c \ ) \mid (\forall \ b \in B) \ \text{ either } a \ r \ b \ \text{ or } b \ s \ c \ \text{ or both}\}$$

The relative addition operator is denoted by the symbol †, pronounced as 'dagger', between the two relation names.

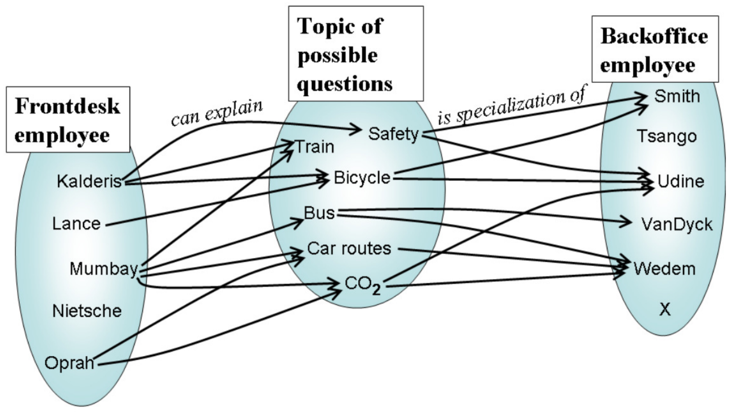

Figure 3.7: Relative addition for Frontoffice-Backoffice teams

To illustrate how relative addition works, consider a company that receives a lot of customer questions about a number of topics. Frontdesk employees are able to explain some of the topics over the telephone to customers. Or they may connect a customer through to some back office employee who specializes in certain topics. A frontdesk employee can be teamed up with a backoffice employee if together, they cover all possible topics. Figure 3.7 shows an instance diagram that contains three possible teams.

The natural language interpretation of relative addition is just that: a tuple $(\ a, c \ )$ is in the relative addition if a and c together cover the entire middle ground of B.

The *relative implication* of r and s, is the relation with signature $(\ r \ [\ s \ )$: $A \ \times \ C$. By definition, its content is *relative implication*

$$\{ \ (\ a, c \ ) \mid (\forall \ b \in B) \ \text{ if } \ a \ r \ b \ \text{ then } b \ s \ c \ \}$$

We denote this operator by a square bracket [, resembling the inclusion symbol $\subset$.

As an example, suppose the institute offers tours abroad for students. Whether a student is eligible for a tour destination, depends on whether he or she has passed for the required courses. Two relations are involved:

– Course *requiredFor* Destination

– Student *passedTheExamFor* Course

and these are combined to produce a new relation

– Student *isEligibleFor* Destination

Figure 3.8 shows a sample of the instance diagram. Caroline is eligible to go to Madrid, because she has passed for the required course Spanish Medieval Literature. For the trip to Rome, two courses are required, Latin and World Religions. Student Ang has passed for both courses, so he qualifies. Student Brown is less fortunate, as she has not yet passed for Latin which is one of the required subjects.

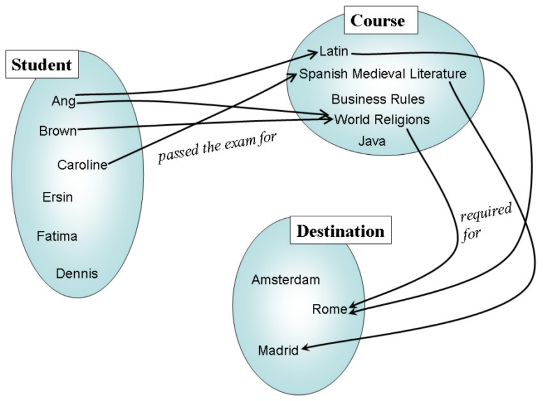

Figure 3.8: Relative addition of *passedTheExamFor* and *requiredFor*

The demand means that for a destination to be open to a student, it is required that for all courses, if Course *requiredFor* Destination, then Student *passedTheExamFor* Course. In other words, for a student to be eligible for a destination, it is required for every course that *passedTheExamFor* [*requiredFor*. Remark that the destination of Amsterdam requires no courses at all, hence all students are eligible irrespective of their achievements in exams.

relative subsumption Finally, the *relative subsumption* of relations r and s, is the relation with signature $(r \rbrack s) : A \times C$. By definition, its content is

$$\{ (a, c) \mid (\forall\, b \in B) \text{ if } b\ s\ c \text{ then } a\ r\ b \}$$

This operator is denoted by the] symbol, which resembles the inclusion in the other direction, for the same reason as above.

3.4 Laws for operations on relations

Using clauses like 'and', 'or', 'if', 'then', 'for all' and 'exists', you can combine simple sentences into rather complex phrases. Just so, relation operators enable you to write complex formulas. But often, one meaning can be phrased in several equivalent ways, and a mathematical formula can be written in several ways. It can be difficult to decide whether different expressions have the exact same meaning, and if they will be true (or false) in exactly the same circumstances. For example:

– $\mathit{for}^{\smile};\mathit{sent}\ \vdash\ \mathit{deliveredTo}$, meaning: 'An Invoice *for* a certain Order is *sent* to a particular Customer, only if that certain Order was *delivererdTo* that particular Customer.' In other words, only invoices for delivered orders should be sent to the customer.

– $\overline{\mathit{deliveredTo}};\mathit{sent}^{\smile}\ \vdash\ \overline{for^{\smile}}$, meaning: 'If an Order was not *delivered to* some Customer and that Customer was *sent* an Invoice, then certainly that particular *Invoice* was not *for* that *Order*.'

Do these formulas express the same business meaning, or not? As a designer, you want to pick an easy way to express a certain text. To do that, you need the ability to rewrite one formula into another without loss of meaning.

This section introduces important laws from Relation Algebra that allows you to do exactly that. The laws are useful, because they allow you to manipulate with entire extensions at once, instead of having to consider all the tuples one by one. As a rule designer, you must learn to use these laws, and manipulate and reason with formulas and operators, just like you have once learned to manipulate with numbers.

3.4.1 Laws for set operators

Operators on sets follow some simple laws. For instance, when determining the intersection of multiple sets, it makes no difference in which order the intersections are calculated. To put it more formally: intersection is

$$\text{associative: } (A\cap B)\cap C = A\cap(B\cap C) = A\cap B\cap C \qquad (3.1)$$

$$\text{commutative: } A\cap B = B\cap A \qquad (3.2)$$

Union, like intersection, is also *associative* and *commutative*, i.e. ∩ may be replaced by ∪ in the two laws above. When a formula combines union and intersection, the laws of *distributivity* apply:

associative
commutative
distributivity

$$\begin{aligned} A\cap(B\cup C) &= (A\cap B)\cup(A\cap C)\\ A\cup(B\cap C) &= (A\cup B)\cap(A\cup C)\end{aligned} \qquad (3.3)$$

The first line reads: “the union with an intersection equals the intersection of the unions”, and the second one can be pronounced as “intersection with a union equals the union of the intersections”.

3.4.2 Laws for composition and relative addition

Composition and relative addition are associative operators. Just like in law 3.1, it does not matter how brackets are placed in an expression with two or more of the same operators. As a consequence, brackets may be omitted and the expression looks less cluttered.

$$(p;q);r \;=\; p;(q;r) \;=\; p;q;r \tag{3.4}$$
$$(p\dagger q)\dagger r \;=\; p\dagger(q\dagger r) \;=\; p\dagger q\dagger r \tag{3.5}$$

neutral

The next law is special for composition. It states that the identity relation $\mathbb{I}$ may be removed if used in a composition: the identity relation is *neutral* with respect to composition and can be omitted.

$$r;\mathbb{I} \;=\; \mathbb{I};r \;=\; r \tag{3.6}$$

diversity

Similarly, you may remove the complement of identity $\overline{\mathbb{I}}$ (sometimes referred to as *diversity*), if it appears in a relative addition.

$$r\dagger\overline{\mathbb{I}} \;=\; \overline{\mathbb{I}}\dagger r \;=\; r \tag{3.7}$$

3.4.3 Laws for the inverse operator

The following laws show how the conversion operator behaves.

$$\text{involutive: } r^{\smile\smile} \;=\; r \tag{3.8}$$
$$\text{distributive: } (r\cup q)^{\smile} \;=\; r^{\smile}\cup q^{\smile} \tag{3.9}$$
$$\text{and also: } (r\cap q)^{\smile} \;=\; r^{\smile}\cap q^{\smile} \tag{3.10}$$

In the next two laws, the order of the relations r and s is reversed:

$$(r;s)^{\smile} \;=\; s^{\smile};r^{\smile} \tag{3.11}$$
$$(r\dagger s)^{\smile} \;=\; s^{\smile}\dagger r^{\smile} \tag{3.12}$$

3.4.4 Laws for distribution in compositions

When working with compositions of relations, it often happens that a union or intersection appears somewhere in the formula. There is one distributive law that allows to remove the union operator from compositions of relations.

$$p;(q \cup r);s \quad = \quad (p;q;s) \cup (p;r;s) \tag{3.13}$$

It would be nice if the same kind of distribution would hold for intersection, ∩, but alas not. Distribution does not apply for intersections in general. Only if certain conditions are met, does distribution apply for intersections. This will be demonstrated later on.

3.4.5 Laws for complement

The best known law concerning complement is that it is an involutive operator: apply it twice and the original relation is reproduced:

$$\text{involutive: } \overline{\overline{r}} = r \tag{3.14}$$

The mathematician *De Morgan* was first to formulate these laws: *De Morgan*

$$\overline{r} \cup \overline{s} \quad = \quad \overline{r \cap s} \tag{3.15}$$
$$\overline{r} \cap \overline{s} \quad = \quad \overline{r \cup s} \tag{3.16}$$

The laws above are frequently used because they allow the complement operator to 'move around' in your formulas. There are similar laws involving ; and † operators, also named after De Morgan:

$$\overline{r};\overline{s} \quad = \quad \overline{r \dagger s} \tag{3.17}$$
$$\overline{r} \dagger \overline{s} \quad = \quad \overline{r;s} \tag{3.18}$$

Equations 3.16 and 3.18 are collectively known as De Morgan's laws. Lesser known, but almost as important are equivalences which De Morgan called "Theorem K". These will be discussed in the next chapter.

3.4.6 Laws about inclusion

Laws about set inclusion are very important, because inclusion lies at the basis of rule assertions that the next chapter is all about. This section provides some laws about the inclusion operator $\vdash$ or $\subset$. The following assertion is obviously true for relations r and s sharing the same type:

$$(r \cap s) \ \vdash \ r \ \vdash \ (r \cup s) \tag{3.19}$$

Inclusion can be combined with the flip and complement operators, which produces the following equivalences. Beware though that the order of r and s is reversed in the second formula only:

$$r \vdash s \quad \Leftrightarrow \quad r^{\smile} \vdash s^{\smile} \tag{3.20}$$
$$r \vdash s \quad \Leftrightarrow \quad \overline{s} \vdash \overline{r} \tag{3.21}$$

monotonicity

Inclusion also combines easily with operators other than flip and complement, sometimes referred to as the *monotonicity* of inclusion. But beware that the following laws are not equivalences but work in only one direction:

$$r \vdash s \quad \Rightarrow \quad r \cap t \vdash s \cap t \tag{3.22}$$
$$r \vdash s \quad \Rightarrow \quad r \cup t \vdash s \cup t \tag{3.23}$$
$$r \vdash s \quad \Rightarrow \quad r;t \vdash s;t \tag{3.24}$$
$$r \vdash s \quad \Rightarrow \quad t;r \vdash t;s$$
$$r \vdash s \quad \Rightarrow \quad r \dagger t \vdash s \dagger t \tag{3.25}$$
$$r \vdash s \quad \Rightarrow \quad t \dagger r \vdash t \dagger s$$

3.4.7 Operator precedence

To conclude this section, we give some conventions that govern precedence of operators. Just like in arithmatics, where for instance "take the square of" takes precedence over addition. These conventions save brackets and simplifies the writing of formulas with multiple operators. Table 3.6 contains the precedence rules.

3.5 Homogeneous relations

So far, we discussed relations based on Cartesian products with sources and targets arbitrary. But the same concept may be used for both source and target. This section is devoted to exactly such relations, and the special features that they have.

homogeneous

A *homogeneous* relation is a relation for which one concept serves both as source and as target.

expression	precedence rule	to be read as
unary operators $r^{\smile}$ and $\overline{r}$ have highest precedence		
$\overline{r}; q$	flip and complement always take precedence	$(\overline{r}); q$
$\overline{r} \dagger r$	(also if flip or complement appear	$(\overline{r}) \dagger q$
$r^{\smile}; q$	at the righthand side)	$(r^{\smile}); q$
$r^{\smile} \dagger r$		$(r^{\smile}) \dagger q$
; and †, having equal precedence, take precedence over $\cap$ and $\cup$, $=$ and $\vdash$		
$p \cap q; r$	; and † bind stronger than $\cap$ and $\cup$, $=$ and $\vdash$	$p \cap (q; r)$
$p \cap q \dagger r$	(also if ; or † appears at the lefthand side)	$p \cap (q \dagger r)$
$p \cup q; r$		$p \cup (q; r)$
$p \cup q \dagger r$		$p \cup (q \dagger r)$
$\cap$ precedes over $\cup$		
$p \cup q \cap r$	$\cap$ binds stronger than $\cup$	$p \cup (q \cap r)$
$=$ and $\vdash$ have weakest precedence of all		
$p = q \cup r$	$\cup$ and $\cap$ bind stronger than $=$ or $\vdash$	$p = (q \cup r)$
$p = q \cap r$		$p = (q \cap r)$
$p \vdash q \cup r$		$p \vdash (q \cup r)$
$p \vdash q \cap r$		$p \vdash (q \cap r)$

Table 3.6: Precedence of operators

All other relations will be called *heterogeneous* relations: their source and target concepts are different. *heterogeneous*

We already encountered one example of a relation with identical source and target, namely the Identity relation, abbreviated $\mathbb{I}$. Examples of homogeneous relations are easy to think of, once you realize that they correspond to simple sentences that express facts about similar terms. Some examples are Person *isRelatedTo* Person, in arithmetics: Number *isGreaterThan* Number, and in chemistry: Compound *mayDecomposeInto* Compound. Or in software maintenance: Software-change *interferesWith* Software-change. In business administration: Department *isSubordinateTo* Department. And in process design: Use-case *isSubvariantOf* Use-case.

In a conceptual diagram, the homogeneous relations are easy to spot because the relation connects a concept with itself, and there will be an arc pointing back to its origin.

Let us consider a relation $r_{[A,A]}$. By our definition, r is homogeneous as its source and target are the same. We can point out several characteristics that the relation r may, or may not have.

3.5.1 Reflexive

A relation $r_{[A,A]}$ is called *reflexive* if for all elements x in the set A, *reflexive* the tuple (x, x) is in r. The relation *isACloseFriendOf* is an example: everybody is a close friend of themselves. The condition can be stated as:

$$\mathbb{I} \vdash r$$

irreflexive

The complete opposite is a relation with the property that identical pairs (x, x) are forbidden. Relations with this property are called *irreflexive*. Two irreflexive examples are *isSpectatorOfPerformanceByActor* or *isParentOf*.

In a matrix representation, a reflexive relation will show markings in all cells on the diagonal (and probably in other places as well, but that does not concern us). To contrast, an irreflexive relation may have markings anywhere, except on the diagonal.

A homogeneous relation can, but need not be reflexive or irreflexive. In general, no specific conditions apply for tuples (x, x) to be, or not to be in the extension. For instance, when elections are held, then a relation Person *votesFor* Person may, or may not show that some people voted for themselves. Of course, the votes remain secret in general.

3.5.2 Symmetric

symmetric

Relation r is called *symmetric* if for any tuple $(x, y) \in r$, the inverse tuple (y, x) is also in r. This can also be written as:

$$r^{\smile} \vdash r$$

We leave it to the reader to verify that a relation is symmetric if and only if equality holds, i.e.

$$r^{\smile} = r$$

Examples of relations that are symmetric (or at least ought to be so) are *isMarriedTo*, *isInAMeetingWith*, and *isInTheSameClassAs*.

asymmetric

A relation r is *asymmetric* if for any tuple (x, y) in r, the inverse tuple (y, x) is not in r, which of course can be written as:

$$r^{\smile} \cap r = \emptyset$$

An example of a relation that is asymmetric is *awardsBonusPaymentTo*. Of course, violations may occur: two managers that award bonuses to one another. Remark that an asymmetric relation is automatically irreflexive. Indeed, we do not want a manager to award a bonus to him- or herself.

Like with (ir)reflexivity, many homogeneous relations are neither symmetric nor asymmetric. The relation Website *hasHyperlinkTo* Website is an example: some websites may link to one another, but there is no rule or obligation to do so.

A homogeneous relation may be 'almost' asymmetric, meaning that it would be asymmetric were it not for some reflexive tuples (x, x). Such relations are called *antisymmetric*, and they are characterized as *antisymmetric*

$$r^{\smile} \cap r \ \subset \ \mathbb{I}$$

3.5.3 Property relation

In some situations, a homogeneous relation p may be both symmetric and antisymmetric at the same time. Some careful reasoning shows that such a relation p must satisfy the condition: $p \vdash \mathbb{I}$. We leave it to the reader to verify that both the identity relation and the empty relation $\emptyset$ are symmetric and antisymmetric.

However, there is more to it, if we consider the elements of the source (and target!) set A. We can divide A into two subsets: one subset with the atoms x that do satisfy $(x, x) \in p$, and the subset of all other elements in A that are not in $p : (y, y) \notin p$. The relation p divides the source set in two disjoint subsets. In other words, the relation p can be understood as a simple Yes/No property of atoms of A. Yes, the atom x is in the subset for which (x, x) is in p. Or no, the tuple (y, y) is not in p. For this reason, a homogeneous relation that is symmetric and antisymmetric is sometimes called a *binary property* for A. *binary property*

By the way: there are other ways to specify properties for atoms of A, as will be shown in the chapter on design considerations.

3.5.4 Transitive and intransitive

If a relation is homogeneous, then we may define the composition with itself. In fact, this can be done over and over again, and you can derive many new homogeneous relations in this way. As an example, consider the relation *isCloseTo*. If Utrecht *isCloseTo* Hilversum, and Hilversum *isCloseTo* Almere, then we have Utrecht *isCloseTo* someplace that *isCloseTo* Almere. A relation r is called *transitive* if the composition with itself is contained in the relation itself: *transitive*

$$r; r \vdash r$$

Another example of a transitive relation is the 'subset' relation among sets: if $A \subset B$ and $B \subset C$ then $A \subset C$, and so (excuse the complicated notation!)

$$\subset; \subset \vdash \subset$$

The opposite of transitivity is shown in family-relationships as studied in genealogy. If we have Person *isParentOf* Person, then we can compose the *isParentOf* relation with itself. The composite is in fact the *isGrandparentOf* relation, and a person may not be both parent and grandparent of a child. Hence, *isParentOf* is an example of an *intransitive* relation, meaning:

intransitive

$$r;r \cap r = \emptyset$$

Repeat the composition and you arrive at Person *isGreatgrandparentOf* Person, etcetera. Genealogy now jumps to one generalized relation Person *isAncestorOf* Person, which is a transitive relation. The reader should verify that this particular relation does satisfy the inclusion $r;r \vdash r$, but equality $r;r = r$ does not hold. Again, like reflexivity and symmetry, many homogeneous relations have neither the transitive nor the intransitive property.

3.5.5 Summary of homogeneous properties

The various properties that homogeneous relations may have, can be written using mathematical symbols. If you do not readily grasp the notations, you should check back in order to practice your skills in translating formulas into natural language, and vice versa. The mathematical shorthand aims to capture the simple business facts, no more, no less. For you, there should be no mystery involved.

reflexive	$(\forall a)\ a\ r\ a$	(3.26)
irreflexive	$\neg(\exists a)\ a\ r\ a$	(3.27)
symmetric	$(\forall a,b)\ a\ r\ b \rightarrow b\ r\ a$	(3.28)
asymmetric	$(\forall a,b)\ a\ r\ b \rightarrow \neg(b\ r\ a)$	(3.29)
antisymmetric	$(\forall a,b)\ a\ r\ b \wedge b\ r\ a \rightarrow a=b$	(3.30)
transitive	$(\forall a,b)\ (\exists x)\ (a\ r\ x) \wedge (x\ r\ b) \rightarrow a\ r\ b$	(3.31)
intransitive	$(\forall a,b)\ a\ r\ b \rightarrow (\forall x)\neg(a\ r\ x) \vee \neg(x\ r\ b)$	(3.32)

3.5.6 Structure of a set

There is a close link between the structure of a set, and the properties of a homogeneous relation. If for a set A we have a homogeneous relation p that is reflexive, symmetric and transitive at the same time, this means that the set A can be partitioned. It is possible to point out a number of subsets (also called partitions) A_1, A_2, A_3 ..., up to A_n, such that every element of A is contained in exactly one of the subsets. The intersection of any two subsets is empty, and the union of all subsets equals the original set A.

equivalence relation

The relation is called an *equivalence relation*. Examples of this kind of situation are easy to find: Employee *worksAtTheSameDepartmentAs* Employee, or Student *hasSameGradeAs* Student. An instance diagram of this kind of relation resembles an archipelago of islands: the entire set consists of islands, every element is in one island, and islands do not overlap. This is also called a *partitioning* of the set A. *partitioning*

It is well possible to have more than one equivalence relation (and matching partitioning) for a concept. For example, cars can be partitioned according to color, or age, or mileage, weight, value or any other property.

If for a set A we have a homogeneous relation p that is reflexive, asymmetric and transitive at the same time, this means that the set A has some ordering or hierarchy. The relation is said to be an *ordering relation* or *order*. For example: Employee *isSubordinateTo* Employee, amounting to hierarchy among workers. Or Department *isPartOf* Department, which corresponds to organizational top-down structure. Other examples are set inclusion and ancestor relations. *ordering relation*

An instance diagram of this kind of set resembles a tree with branches, with the elements arranged along the separate branches. Or to be precise: one or more trees. Notice that in general, the ordering is incomplete, as not every possible combination of two employees will participate in the relation: neither is subordinate to the other. In an instance diagram, these employees will appear on separate branches. This is called a *partial order* because there exists tuples (x, y) such that neither $(x, y) \in p$ nor $(y, x) \in p$. *partial order*

It is called a *total order*, or linear order, if any two elements can be compared: either $(x, y) \in p$ or $(y, x) \in p$. In other words, $p \cup p^{\smile} = \mathbb{V}$. For a linear order, the instance diagram will resemble a single line, or a tree with just a single branch. All elements will be neatly arranged on that line. An example is the common *isSmallerThanOrEqualTo* relation on numbers. *total order*

3.5.7 IsA relation

To conclude this section on homogeneous relations, we finish with an ubiquitous relation that is *not* homogeneous. In large sets, one can point out all kinds of subsets. For example, in the set of all customers we can pick out all customers who joined last week, or who haven't paid the last invoice, or who live in New Amsterdam, or who have not ordered a Harry Potter book. The corresponding sentences are very simple, like 'X, the customer who joined last week *is* customer X', or 'Y, the customer who lives in New York *is* customer Y'. The common template looks like '.... *isA* ....' In all cases, we have a relation with the subset as source, and the enveloping set as target. An instance of the former *isA* instance of the latter, no exceptions.

The definition (intension) of the first concept, which is the subset, is more limited, more restrictive than the intension of the enveloping concept. And necessarily, the extension of the first concept is contained in the extension of the other concept. Whenever this is the case, we call the source the *specialisation*, the target being the *generalisation*. *specialisation* *generalisation*

> An *inclusion relation* is a relation for which the source is a proper subset of the target *inclusion relation*

This kind of relation is quite common in conceptual models, and they are easy to spot because they are usually named *isA*. Examples of this kind of relation are easy to think of: Student *isA* Person. Or in arithmetics: Prime-Number *isA* Number. Or chemistry: Pure-Substrate *isA* Compound. But please do not make the mistake to think that these are homogeneous relations, as their sources and targets are really different!

As a closing remark: it is quite possible that the extension of one concept is a subset of the extension of another one. All customers in the shop happen to be male, all company cars run on diesel fuel. But this alone, is not enough for generalisation-specialisation. The customers need not be male by definition; company cars may run on any kind of fuel and still be a company car. Here, the subset property is just a coincidence, and the extensions of the concepts, without changing definitions, can be quite different at some other time.

3.6 Multiplicity constraints

Simple sentences express single facts. By studying the deep structure of sentences, we arrived at the notion of relation, called 'fact-type' by the Business Rules Manifesto. The Manifesto states that rules build on facts. Indeed, even a single relation can be subjected to control: the relation must satisfy some business rule. A business rule that governs a single relation is by its very nature rather simple and easy to understand. Such rules are frequently encountered, and they have their own name: *multiplicity constraint*s. They express knowledge about how the tuples in a relation should be organised, and what behaviour is expected or prevented.

multiplicity constraint

From mathematics we learn that multiplicity constraints come in exactly four flavours. It is the duty of the rule analyst to establish for every relation which constraints apply. As the four flavours can be combined in any way, a total of 16 combinations is possible. Indeed, relations can be found in practice where no constraints apply and anything goes, up to relations that are under very restrictive multiplicity constraints.

3.6.1 Univalent and Total

These two multiplicity constraints must be verified at the source concept. A relation r is said to be:

univalent

- *univalent*, if every atom in the source occurs in at most one tuple of the relation, or: every atom in the source is related to at most one atom in the target,

total

- *total*, if every atom in the source always occurs in at least one tuple of the relation, or: every atom in de source is related to at least one atom in the target.

These properties are easily verified in a matrix representation of the relation. If we write the relation with the source at the left, and the target across, then univalent means that there is at most one hit on every row. Total means that there is at least one hit on every row, and more than one hit is also fine.

In an instance diagram (figure 3.9), univalence means that at most one arc emanates from every source element. Total means that at least one outgoing arc is drawn for each element.

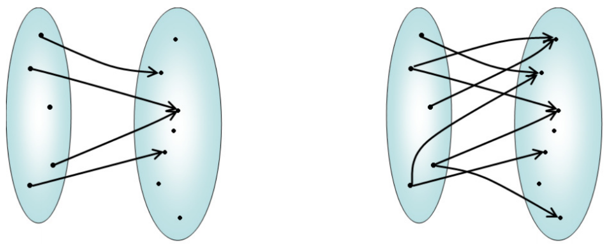
Figure 3.9: Left: univalent, not total; right: total, not univalent

3.6.2 Injective and Surjective

These two multiplicity constraints are to be verified at the target. The relation r is called

- *injective*, if every atom in the target occurs in at most one tuple of the relation, or: every atom in the target is related to at most one atom in the source. *injective*
- *surjective*, if every atom in the target always occurs in at least one tuple of the relation, or: every atom in the target is related to at least one atom in the source. *surjective*

Again, these properties are easy to verify in a matrix representation. Source at the left, target across, then an injective relation will show at most one hit in each column, whereas a surjective relation will show one or more hits in every column of the matrix.

In an instance diagram (figure 3.10), injective means that in each target element, at most one arc arrives. Surjective means that at least one incoming arc is drawn to each element.

3.6.3 Function

Some combinations of multiplicity constraints appear very frequently in practical applications, and they deserve their own name. In particular:

- a relation r is a *function* if it is both *univalent* and *total*: every atom in the source is related to exactly one atom in the target concept. *function*

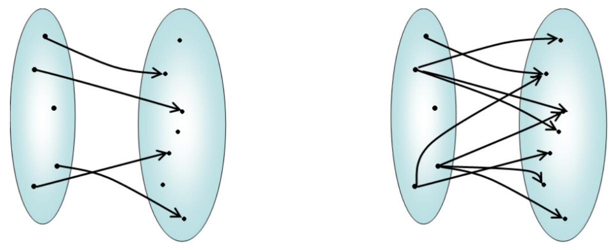

Figure 3.10: Left: injective, not surjective; right: surjective, not injective

To emphasize the property of being a function, we often use a shorthand notation. We simply write

$$r : A \to B$$

the arrow here means that relation r has signature $r_{[A,B]}$ and it is both univalent and total. Alas, the meaning of this arrow in the formula, differs from the meaning of arrows used in Conceptual Models such as figure 3.4 or Instance Diagrams such as figure 3.11.

The word 'function' is also used in mathematics, with notation $f(x) = y$. For instance the function 'square': $f(x) = x^2$. It is no coincidence that the same word is used: in a mathematical function, each and every number x is related to exactly one other number y. Thus, the mathematical function 'square' is both univalent and total. This becomes clear if we write (x, x^2) instead of $f(x) = x^2$. This relation contains tuples such as $(1, 1)$, $(2, 4)$, $(3, 9)$ but also $(0, 0)$, $(-1, 1)$, $(-2, 4)$. Indeed, the 'square' function produces correct tuples in this way, exactly one tuple for each number x.

3.6.4 Injection, Surjection, Bijection

Functions are frequently encountered, and they often even have additional multiplicity constraints, either injectivity or surjectivity. A function r (a relation that is both univalent and total) is called:

injection
surjection
bijection

- *injection*, if the function is injective
- *surjection*, if the function is surjective
- *bijection*, if it is both injective and surjective.

Figure 3.11 depicts these properties. Notice in the rightmost example that the inverse relation is also a function. A closer inspection reveals that a bijection requires that the source and target datasets must have exactly the same number of atoms, for each atom in the source is related to one target element, and reversely.

Although most relations are subject to some multiplicity constraints, this does not mean that such rules apply for every concept in every relation. For example, a business may dictate that each invoice is sent to one customer. Or, for every invoice in the source concept, there is exactly one tuple in the relation *isSentTo*

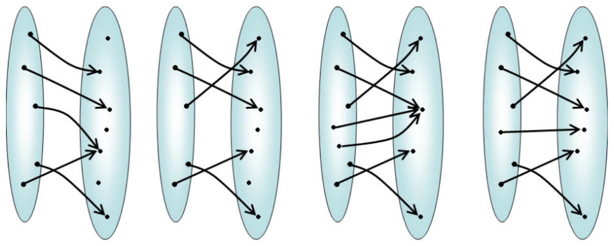

Figure 3.11: Common combinations of multiplicities. Left to right: function, injection, surjection, and bijection.

in which that invoice appears. But not the other way around: one customer may be sent no invoice at all, or she may receive hunderds.

Multiplicity constraints are very important in conceptual models, and we will discuss some more features of multiplicities in the next chapter that is all about rules.