CC Primer

S. Joosten

January 13, 2004

Abstract

Creating useful and relevant specifications becomes easier, once you know how to write rules in a precise manner. The CC-technique helps to formulate rules correctly, to describe design patterns using these rules and to assemble specifications from design patterns.

This text introduces relations from the very beginning, assuming only a basic understanding of sets. By applying relational calculus to real world problems, this paper develops a powerful tool for designers of information systems.

Keywords: Specifications, Relational Calculus, Design patterns, Conceptual modelling.

ISRL Categories: CB0901.03 Modelling languages, etc.

1 Introduction

Professionals in information systems deal with large collections of data, such as a telephone directory, an accounts administration, student records, etcetera. In their seminal works about relational calculus, mathematicians of the 19th century, such as De Morgan, Peirce, and Schröder have provided us with the means to do just that [1, 4, 6]. More than a century ago, they have enabled the professionals of today to describe and manipulate data effectively, quickly, and without mistakes.

Precisely these skills will allow you to specify large information systems. A proper understanding of relations, combined with your ability to manipulate them, is a valuable tool to analyze problems and to invent feasible, simple solutions. It will improve your specifications and it will make you a better designer. If you are that professional, who wants to do a proper

conceptual analysis, or the engineer who wants to make useful and concrete specifications, this text is meant for you.

Specifications in the CC-technique consist of rules. Rules make things specific, which is the essence of a *specific*ation. Once you are familiar with CC, you will use rules in practical situations to represent whatever must be specified. CC was designed to make conceptual analysis as easy and precise as calculating with numbers. Since rules consist of concepts and relations, you will be calculating with concepts and relations rather than numbers. Relational calculus[1] is the name of the game. Assuming that this is a whole new ballgame for you, I have designed this primer for professionals who want to learn as much formalism as required and not a single bit more. There are numerous textbooks, readers, and scientific literature about relational calculus. In fact, many excellent introductions exist to date. One of my favourites is the "Background Material" of Jipsen, Brink, and Schmidt[2] [3]. So why write yet another introduction? In the first place, I have limited this introduction to a bare minimum. Besides, I have learned from my students that a strong link to practice is needed. Most of them feel unfamiliar with the theory, even though they know about relational databases. They want a starter's manual before picking up a reference manual. Whether my attempt to make things easier works, is left for you to find out.

2 Preliminaries

In order to define relations, we need sets. Assuming you are familiar with the idea of sets, I will merely introduce set notations and rules that are used in the sequel. As a reading aid, I will stick to the following typographic conventions: Sets are written with names starting with a capital letter, such as $Book$, A, and PDC. The name itself (e.g. $Book$) may provide you with a clue as to which kind of elements are in the set. Lowercase letters, possibly suffixed by apostrophes, denote these elements. Where appropriate, such a lowercase letter will match the first letter of the set.

The symbols $b \in Book$ express the fact that b is an element (or member) of the set $Book$.

Given a set X and a property P, the set of all elements of X that satisfy P is denoted by:

$$\{x \in X |\ P(x)\} \qquad or \qquad \{x | x \in X \text{ and } P(x)\} \tag{1}$$

[1] to be precise: Heterogeneous Relation Algebra [5]

[2] In this introduction, the reader may recognize some of the skillfully crafted sentences and formulas from Jipsen, Brink, and Schmidt, whose work I gratefully acknowledge.

The property P is often described informally, but is understood to be an abbreviation for a precise expression in some formal language.

We will use the following relations and operations on sets throughout this paper:

- *equality:* $X = Y$ if X and Y contain the same elements,
- *inclusion:* $X \subseteq Y$ if every element of X is also an element of Y,
- *union:* $X \cup Y$ is the set of elements in X or Y,
- *intersection:* $X \cap Y$ is the set of elements in X and Y,

These notations allow us to manipulate with sets in a formal way, using the well-established rules. The main ones are captured by equations 2 through 10. For any set X, Y, and Z

$$X \subseteq X \qquad (\subseteq \text{ is reflexive}) \tag{2}$$
$$X \subseteq Y \text{ and } Y \subseteq Z \text{ imply } X \subseteq Z \qquad (\subseteq \text{ is transitive}) \tag{3}$$
$$X \subseteq Y \text{ and } Y \subseteq X \text{ imply } X = Y \qquad (\subseteq \text{ is antisymmetric}) \tag{4}$$
$$X \cup Y = Y \cup X \qquad (\cup \text{ is commutative}) \tag{5}$$
$$X \cap Y = Y \cap X \qquad (\cap \text{ is commutative}) \tag{6}$$
$$(X \cup Y) \cup Z = X \cup (Y \cup Z) \qquad (\cup \text{ is associative}) \tag{7}$$
$$(X \cap Y) \cap Z = X \cap (Y \cap Z) \qquad (\cap \text{ is associative}) \tag{8}$$
$$(X \cap Y) \cup Z = (X \cup Z) \cap (Y \cup Z) \qquad (\cup \text{ distributes over } \cap) \tag{9}$$
$$(X \cup Y) \cap Z = (X \cap Z) \cup (Y \cap Z) \qquad (\cap \text{ distributes over } \cup) \tag{10}$$

Such rules are useful, because they allow you to manipulate with sets without any reference to their contents.

3 Relations

Relations are the basic entity, out of which specifications are built. Think of table 1 as an example of a relation that shows which client has paid which invoice. Let us call this relation *paid*. It actually stands for three statements: Client Applegate has paid invoice number 5362a, Brown has paid invoice 721i, and Conway has paid invoice 9443a. Larger and more realistic examples, such as a telephone directory, are impractical to quote entirely. In this text you will learn to work with relations independent of their contents.

Client	*Invoice*
Applegate	*5362a*
Brown	*7̸21i*
Conway	*9443a*

Table 1: Contents of *paid*

Any relation has a *name*, a *left attribute*, a *right attribute*, and *contents*. By convention, names of relations start with a lowercase letter. Names of concepts (i.e. left- and right attributes) start with an uppercase letter. For example, the relation in table 1 bears the name *paid*. We call *Client* the left attribute of this relation and *Invoice* the right attribute. In fact, both *Client* and *Invoice* are sets of which you will find elements in the corresponding columns.

Together, the name, left attribute and right attribute are called the *signature*, which we write as:

$$paid : Client \times Invoice \tag{11}$$

The signature identifies a relation uniquely. That is: if two relations have identical names, left- and right attributes, they have the same contents. As a consequence, different relations may have identical names, provided one or both attributes are different. So name and attributes determine a relation uniquely. When confusion is impossible, we will refer to a relation by its name only. In other cases, we will refer to the relation as $R_{[A\times B]}$.

Actually, the notation $A \times B$ means the *cartesian product* of A and B. That is: $A \times B$ is the set of all pairs of which the left element is an element of A and the right element is an element of B:

$$A \times B \;=\; \{\langle a, b\rangle |\; a \in A \wedge b \in B\} \tag{12}$$

The symbols $\langle a, b\rangle$ denote a *pair* with the characteristic property $\langle a, b\rangle = \langle a', b'\rangle$ if and only if $a = a'$ and $b = b'$. As a consequence of definition 12, every relation r that has left attribute A and right attribute B, is a subset of $A \times B$:

$$rel_{[A\times B]} \;\subseteq\; A \times B \tag{13}$$

Instead of $\langle x, y\rangle \in r$, we usually write $x\ r\ y$. If we want to write, for example, that *Applegate* and *5362a* are related in the relation *paid*, we write:

$$Applegate\ paid\ 5362a \tag{14}$$

A relation can be represented in many different ways, such as *mathematical* forms, *tabular* forms, in *matrix* forms, and various *graphical* forms. Let us look at an example. We can define the relation $provided_{[Provider \times Delivery]}$ in a mathematical notation as follows:

$$\begin{array}{lcl} provided & : & Provider \times Delivery \\ provided & = & \{\langle \texttt{Candy's candy}, \texttt{Cookies \#0382} \rangle, \\ & & \langle \texttt{Carter}, \texttt{Jelly beans \#4921} \rangle \quad , \\ & & \langle \texttt{Carter}, \texttt{Peanut butter \#1993} \rangle \ \} \end{array} \tag{15}$$

This relation means that a delivery of cookies, marked with delivery number 0382, has come from Candy's candy, whereas Carter has provided both the jelly beans (delivery number 4921) and the peanut butter (#1993) in two deliveries. Both attributes are sets, which might for instance be defined by:

$$\begin{array}{lcl} Provider & = & \{\texttt{Candy's candy}, \texttt{Carter}, \texttt{Walmart}\} \\ Delivery & = & \{\texttt{Cookies \#0382} \quad , \\ & & \texttt{Jelly beans \#4921} \ , \\ & & \texttt{Peanut butter \#1993}\} \end{array} \tag{16}$$

Example (15) represents the relation *provided* in a mathematical form. A tabular representation of the same relation looks like this:

Provider	**Delivery**
`Candy's candy`	`Cookies #0382`
`Carter`	`Jelly beans #4921`
`Carter`	`Peanut butter #1993`

(17)

A matrix representation looks like this:

	`Candy's candy`	`Carter`	`Walmart`
`Cookies #0382`	×		
`Jelly beans #4921`		×	
`Peanut butter #1993`		×	

(18)

A Venn-diagram is a graphical form of representing a relation, which is shown in figure 1. Venn-diagrams show the contents of a relation and the left attribute and right attributes.

In practice, people write relations in many different and even unique forms. A telephone directory relates names and addresses to telephone numbers in a tabular manner and a dictionary does a similar thing with words and their

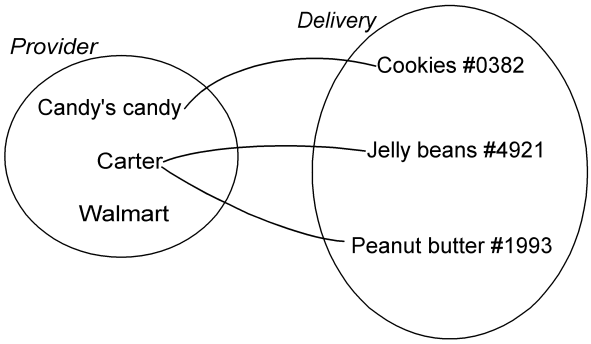

Figure 1: Graphical representation of *provided*

meanings. Still, the representations of telephone directories and dictonaries may vary, even though both are in essence tabular forms. The effort people spend in communicating the contents of such relations underlines the importance of carefully choosing your representations. As a designer, you too will spend that effort once you expose the contents of a relation to the public. Before that time, however, you will work with relations without the slightest reference to their contents.

By abstracting away from the contents, you can oversee the structure of many relations at the same time. For that purpose, you may find it useful to draw *conceptual diagrams*. Figure 2 contains an example, which repre-

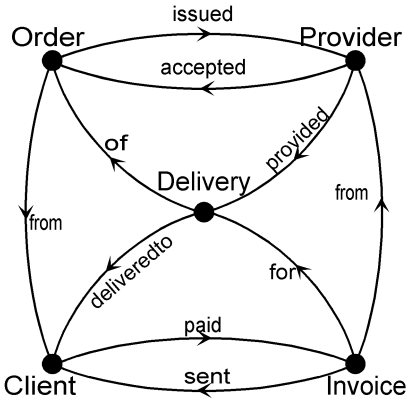

Figure 2: conceptual model of trading

sents ten relations in one diagram. In conceptual diagrams, arcs represent signatures. Each dot in the diagram is called a *concept*. Concepts repre-

sent sets, which are used as attributes in relations. The name of a relation is written near the arc, located such that it is clear which name belongs to which arc. A little arrowhead in the middle of an arc points from left attribute to right attribute. Thus, each arc represents precisely one signature. The little arrowhead helps you to remember the order of attributes. Note that each line determines a corresponding relation uniquely, because the complete signature, that is name, left attribute and right attribute, is determined by that single line and the signature identifies the contents (as explained on page 4).

Figure 2 represents the following signatures:

$$\begin{aligned}
\mathit{paid} &: \mathit{Client} \times \mathit{Invoice} && (19)\\
\mathit{sent} &: \mathit{Invoice} \times \mathit{Client} && (20)\\
\mathit{deliveredto} &: \mathit{Delivery} \times \mathit{Client} && (21)\\
\mathit{for} &: \mathit{Invoice} \times \mathit{Delivery} && (22)\\
\mathit{from} &: \mathit{Order} \times \mathit{Client} && (23)\\
\mathit{from} &: \mathit{Invoice} \times \mathit{Provider} && (24)\\
\mathit{provided} &: \mathit{Delivery} \times \mathit{Provider} && (25)\\
\mathit{of} &: \mathit{Delivery} \times \mathit{Order} && (26)\\
\mathit{accepted} &: \mathit{Provider} \times \mathit{Order} && (27)\\
\mathit{issued} &: \mathit{Order} \times \mathit{Provider} && (28)
\end{aligned}$$

In order to build specifications using rules, we need a vocabulary that allows us to write those rules. This vocabulary consists of relations and operators. You will use relations as your basic building blocks and operators as mortar to combine relations into rules that describe accurately whatever you want to specify.

Set operators are already a part of the vocabulary of relations, since relations are sets themselves. So the expression $r \cup s$ is the set of pairs in r or in s and $r \cap s$ is the set of pairs in r and in s. Similarly, the expression $r \subseteq s$ means that every pair in r is also a pair in s.

Set operators alone will not do, however. We need new vocabulary as well, most notably the *converse* of a relation, the *composition* of two relations, and the *identity relation*.

- *converse:* Let $r : A \times B$, then $\sim r$ is a relation, which is defined by:

$$\sim r : B \times A \qquad\qquad \sim r = \{\langle b, a\rangle |\ a\ r\ b\} \qquad (29)$$

The converse of a relation swaps the left column and the right column in the tabular form. It mirrors the matrix form of a relation along its diagonal.

- *composition :* Let $r : A \times B$ and $s : B \times C$, then $(r; s)$ is a relation, which is defined by:

$$\begin{array}{rcl} (r;s) & : & A \times C \\ r;s & = & \{\langle a, c\rangle | \text{ there exist } b \text{ such that } a\ r\ b \text{ and } b\ s\ c\} \end{array} \quad (30)$$

  If you know about relational databases, you might like to know that composition corresponds to the natural join operator.

- *identity relation:* For every concept C, the identity relation id_C is defined by:

$$id_C : C \times C \qquad\qquad id_C = \{\langle c, c\rangle |\ c \in C\} \quad (31)$$

Let us look at some examples, based on *provided* (relation 15 on page 5), *paid* (relation 1 on page 4), and the relation $for_{[Invoice \times Delivery]}$, which is defined by:

$$\begin{array}{rcl} for & : & Invoice \times Delivery \\ for & = & \{\langle \texttt{721i}, \texttt{Cookies \#0382}\rangle \quad , \\ & & \ \langle \texttt{5362a}, \texttt{Jelly beans \#4921}\rangle \quad , \\ & & \ \langle \texttt{9443a}, \texttt{Peanut butter \#1993}\rangle\} \end{array} \quad (32)$$

The converse of *for* is:

$$\begin{array}{rcl} \sim for & : & Delivery \times Invoice \\ \sim for & = & \{\langle \texttt{Cookies \#0382}, \texttt{721i}\rangle \quad , \\ & & \ \langle \texttt{Jelly beans \#4921}, \texttt{5362a}\rangle \quad , \\ & & \ \langle \texttt{Peanut butter \#1993}, \texttt{9443a}\rangle\} \end{array} \quad (33)$$

The composition of *paid* and *for* is obtained by applying definition 30. Figure 3 illustrates this composition.

$$\begin{array}{rcl} paid; for & : & Client \times Delivery \\ paid; for & = & \{\langle \texttt{Brown}, \texttt{Cookies \#0382}\rangle \quad , \\ & & \ \langle \texttt{Applegate}, \texttt{Jelly beans \#4921}\rangle, \\ & & \ \langle \texttt{Conway}, \texttt{Peanut butter \#1993}\rangle\ \} \end{array} \quad (34)$$

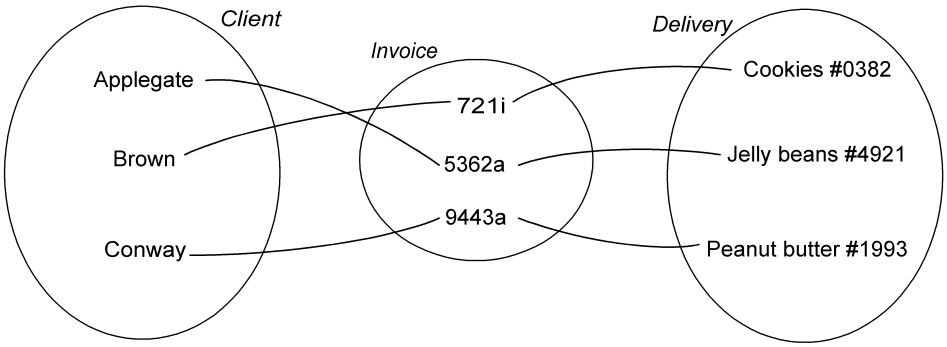

Figure 3: composition of *paid* and *for* (see definition 30)

You cannot compose just any relation. The relation $provided; for$ is not defined, because definition 30 requires the right attribute of *provided* to be the same as the left attribute of *for*. *Delivery*, being the right attribute of *provided*, is clearly not the same as *Invoice*, which is the left attribute of *for*. Nevertheless, composing *provided* with $\sim for$ is defined, and the result can be obtained by applying definition 30:

$$\begin{array}{lcl} provided; \sim for & : & Provider \times Invoice \\ provided; \sim for & = & \{\langle \texttt{Brown}, \texttt{Cookies \#0382} \rangle \quad , \\ & & \langle \texttt{Applegate}, \texttt{Jelly beans \#4921} \rangle, \\ & & \langle \texttt{Conway}, \texttt{Peanut butter \#1993} \rangle \ \} \end{array} \tag{35}$$

Having introduced relations and defined operators on them, let us turn to rules.

3.1 Rules

Rules are used to specify your business, whatever that business might be. Useful rules allow you to talk to various stakeholders in their own language. In practice, any stakeholder may propose rules. As a designer, you will make rule proposals concrete and propose new rules that others have not yet thought of. The CC-technique lets you represent rules in relational calculus, making each one of them concrete enough to expose their violations. It also lets you discover new rules to propose to the business. Any rule proposal can be submitted to a body that is authorized to impose new rules. As a designer, you rarely have that authority. It is however your responsibility as a designer to make rules explicit, to expose possible inconsistencies, and to present rules in a way that stakeholders in the business can understand.

Let us elaborate some examples in the context of figure 2. In that situation, clients and providers are trading according to a simple pattern of orders, deliveries and invoices. The figure proper does not contain any rules; it is merely a picture in which ten different relations are represented by their signatures. In this section, I propose six rules. Each rule is represented in relational calculus, emphasizing the line of reasoning that brings us from informal text to a precise formula.

For starters, how do we express that a provider does not accept any order issued to someone other than himself? Relation $issued_{[Order \times Provider]}$ in figure 2 apparently tells us which orders were issued to which providers. Relation $accepted_{[Provider \times Order]}$ states which providers have accepted which orders. Our first rule states that providers need not accept all orders; they are supposed to accept only those orders that were issued to them. So the set of orders that have been accepted by a provider is a subset of the orders that were issued to him. In other words, every element $\langle p, o \rangle$ of the relation *accepted* is also an element of $\sim$ *issued*. Recalling the definition of inclusion on page 3, we can write this rule as:

$$accepted \subseteq \sim issued \tag{36}$$

Rule 36 uses relations to represent that a provider does not accept an order issued to someone other than himself. This is exactly what we meant to represent, so here we have a first example of a rule that represents business knowledge.

Having successfully represented a first rule, let us try another one. How can we state that no client should ever pay an invoice that was not addressed to him? First, we establish that this rule involves the relations *sent* and *paid*. Then we reformulate the rule, stating that invoices paid by a client can be only those invoices that are sent to that client. In other words, every element $\langle i, c \rangle$ of the relation *paid* is also an element of $\sim$ *sent*. This is represented by:

$$paid \subseteq \sim sent \tag{37}$$

This rule is quite similar to rule 36, because the reasoning that leads to this rule is quite similar. An important use of these rules is to identify violations explicitly. For instance, any invoice that is paid by a client but not sent to that client constitutes a violation of rule 37. Any invoice sent to one client but paid by another client also constitutes a violation. However, an invoice that was sent but not paid does not yield a violation. This is precisely what we want, because it might be still in the mail or waiting on a desk.

Let us see if other rules can be identified. How can we describe that no delivery is provided without an order? First, let us establish that this rule involves the relations *provided*, *accepted*, and *of*. The relation *provided* states which provider has provided which delivery. Relation *accepted* says which provider has accepted which order, and *of* relates a delivery to an order. If an element $\langle p, d \rangle$ is an element of the relation *provided*, then it must also be an element of a relation that says that there is an order, which was accepted by the provider and upon which the delivery was made. This rule is written as:

$$provided \subseteq accepted; \sim of \tag{38}$$

Let us verify that this is exactly the rule we want. The second part of this rule, $accepted; \sim of$, represents the relation that contains pairs $\langle p, d \rangle$ of $Provider \times Delivery$ for which there exist $o \in Order$ such that p *accepted* o and o ($\sim$ *of*) d. You may verify this by checking on definition 30. Note that o ($\sim$ *of*) d is equivalent to d *of* o (by definition 29). A violation of this rule occurs every time a delivery is made without an order. So, this third rule represents precisely what we intended.

The client however, will not accept invoices for orders undelivered. Can we represent that in a fourth rule? Apparently, this involves the relations *sent*, *for*, and *deliveredto*. Relation *sent* tells us which invoice was sent to which client; relation *for* tells us for which delivery the invoice was made; and relation *deliveredto* specifies which delivery was delivered to which client. If there is an invoice for delivery d, which was sent to client c, then $\langle d, c \rangle$ must be in *deliveredto*. Therefore, we can write:

$$\sim for; sent \subseteq deliveredto \tag{39}$$

This rule is useful to expose violations just like the previous ones. If an invoice is sent to a client different from the one who received the delivery, we have a violation. If an invoice is about a delivery that was never delivered, that too constitutes a violation. If there is no invoice, there is no violation of this particular rule. It might still be in the mail, after all.

After discussing four rules, let us look back at figure 2. You may have noticed that every time we discussed a rule, the relations involved in that rule form a cycle in figure 2. This is no coincidence. Each rule has a relation in the left hand side and a relation on the right hand side. These two relations form two different paths in figure 2 between the same concepts. Hence, the relations involved form a cycle, which encloses an area in figure 2.

By turning this principle around, we can use it to discover new rules. For instance, let us investigate the relations $from_{[Invoice \times Provider]}$, for, and $provided$. These three relations form a cycle in figure 2, which has not been covered by any of the previous rules. Can we propose a rule, using these three relations? If there is an invoice for delivery d that comes from provider p, then $\langle p, d \rangle$ must be in $provided$. One part of this rule, which says whether there is an invoice for delivery d that comes from provider p, is represented by $\sim from; for$ (by definition 30). In relational terms, we are saying that any pair $\langle p, d \rangle$ that is in $\sim from; for$ is also in $provided$. However, the other way around is also useful: for any pair $\langle p, d \rangle$ that is in $provided$ there must be an invoice for delivery d that comes from provider p. The rule works in two directions: the relations $provided$ and $\sim from; for$ are a subset of one another. Using property 4, we get:

$$provided \;=\; \sim from; for \tag{40}$$

Reflecting on this result, you may have noticed that this rule was discovered merely by following the procedure of inspecting relations in a cycle. There was no flash of ingenuity, no spectacular discovery involved. Also notice that equation 40 can be used to define the relation $provided$. If relations $from$ and for are known, the contents of $provided$ can be computed from their contents.

Each of the previous rules 36 through 39 might have been discovered in a similar way, systematically inspecting the cycles in figure 2. In fact, most areas in figure 2 are enclosed by the cycles discussed so far. Only the area enclosed by relations $from_{[Order \times Client]}$, of and $deliveredto$ remains to be analyzed. So these three relations are a candidate for a rule. By similar reasoning as in the previous rule, our proposal will be:

$$deliveredto \;=\; of; from \tag{41}$$

This rule says: if a pair $\langle d, c \rangle$ is in $deliveredto$, there must be an order for delivery d that comes from provider p. Reversely, if there is an order for delivery d that comes from provider p, $\langle d, c \rangle$ must be in $deliveredto$. So this rule too is a proper candidate for agreement among stakeholders in the business.

Having presented six rules, I hope that you have acquired the idea of inventing rules by chasing cycles. It is a skill, basic to the CC-technique. This skill is fundamental in analyzing the rule proposals of various stakeholders and making these proposals concrete.

3.2 Combining rules

Let us pursue the following idea: if larger cycles are made up from smaller ones, and if cycles correspond to rules, can we use that to create new rules? Can we take two rules that correspond to adjacent areas and combine them to form a new rule?

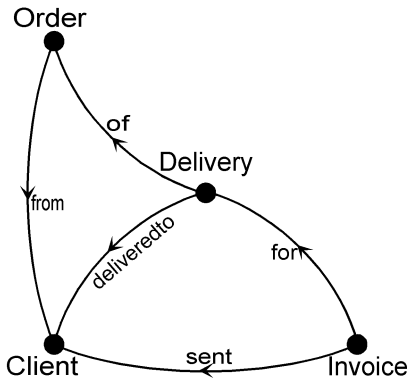

Figure 4: conceptual model of trading

For example, the cycles that correspond to rules 39 and 41, drawn again in figure 4, form adjacent areas. The relation *deliveredto* is shared between them. This observation inspires the following derivation:

$$\begin{array}{ll} & \sim for; sent \\ \subseteq & \qquad \{\text{rule } 39\} \\ & deliveredto \\ = & \qquad \{\text{rule } 41\} \\ & of; from \end{array}$$

As a result, we may conclude:

$$\sim for; sent \ \subseteq \ of; from \tag{42}$$

This result says: if there is an invoice for a delivery, which has been sent to some client c, then there is an order for that delivery, which has come from client c. So the idea has worked: adjacent areas in a conceptual diagram may inspire new rules. The new rule can be derived from existing rules, so this new rule cannot be presented to stakeholders as a new rule proposal. If rules 39 and 41 have already been accepted, rule 42 is a logical consequence.

In a similar fashion we can explore other parts of figure 2, and do some more reasoning with rules:

$$\begin{array}{ll} & \sim \mathit{from}; \mathit{for} \\ = & \qquad\qquad \{\text{rule } 40\} \\ & \mathit{provided} \\ \subseteq & \qquad\qquad \{\text{rule } 38\} \\ & \mathit{accepted}; \sim \mathit{of} \\ \subseteq & \qquad\qquad \{\text{rule } 36\} \\ & \sim \mathit{issued}; \sim \mathit{of} \end{array}$$

This derivation says: if there is an invoice for a delivery, that came from some provider p, then there is an order for that delivery, which was issued to provider p. Or, in relational terms:

$$\sim \mathit{from}; \mathit{for} \ \subseteq \sim \mathit{issued}; \sim \mathit{of} \tag{43}$$

3.3 Cardinalities

On many occasions, relations are qualified in terms of phrases like "one-to-many", "surjective", "1..n", and the like. If you have ever wondered what this means, you can check it by means of the definitions in this section. Such properties are called *cardinalities*, because they provide bits of information about the number of items in a relation. Cardinalities occur so often, that each of them has been given a name of their own: univalent, total, function, injective, surjective and bijective. We will first define the cardinalities. Then we will express some common conventions about cardinalities in terms of these definitions.

- **univalent**
  A relation $r : A \times B$ is *univalent* if each element of A corresponds to at most one element of B. This property is defined by:

  $$\sim r; r \ \subseteq \ id_B \tag{44}$$

  The definition says that $a \ r \ b$ and $a \ r \ b'$ imply $b = b'$ for all a, b, and b'. A univalent relation is also called a *partial function*.

- **total**
  A relation $r : A \times B$ is *total* if each element of A corresponds to at least one element of B. This property is defined by:

  $$id_A \ \subseteq \ r; \sim r \tag{45}$$

  The definition says that for all $a \in A$ there exists $b \in B$ such that $a \ r \ b$.

- **function**
  A relation is a *function* if it is both univalent and total. That is: a relation $r : A \times B$ is a *function* if every element of A corresponds to precisely one element of B. If r is not total but univalent then it is called a *partial function*.

- **injective**
  A relation $r : A \times B$ is *injective* if each element of B corresponds to at most one element of A. This property is defined by:

  $$r; \sim r \ \subseteq \ id_A \tag{46}$$

  The definition says that $b \ r \ a$ and $b \ r \ a'$ imply $a = a'$ for all a, a', and b. Notice that the property injective is defined conversely to the property univalent.

- **surjective**
  A relation $r : A \times B$ is *surjective* if each element of B corresponds to at least one element of A. This property is defined by:

  $$id_B \ \subseteq \sim r; r \tag{47}$$

  The definition says that for all $b \in B$ there exists $a \in A$ such that $a \ r \ b$. Notice that the property surjective is defined conversely to the property total.

- **bijective**
  A relation is *bijective* if it is a function that is both injective and surjective. That is: a relation $r : A \times B$ is a *bijective* if every element of A corresponds to precisely one element of B and every element of B corresponds to precisely one element of A.

Database designers are familiar with cardinalities, because these properties have significant impact on implementation choices. Figure 5 shows two popular ways in which cardinalities are shown in database models. The left hand side shows the convention in UML [2]. On the right, you can see the "crow's foot" notation, which has become popular in entity relationship modelling (e.g. James Martin). Even though cardinality notations may differ from one graphical modelling technique to another, they all come down to the same set of properties. If you are using graphical modeling techniques, you may want to write the name of a relation near its right attribute, so you can easily remember the signature of that relation. This is of course not necessary in conceptual diagrams, as shown in figure 2, because the little arrowhead in the middle disambiguates the signature of every relation.

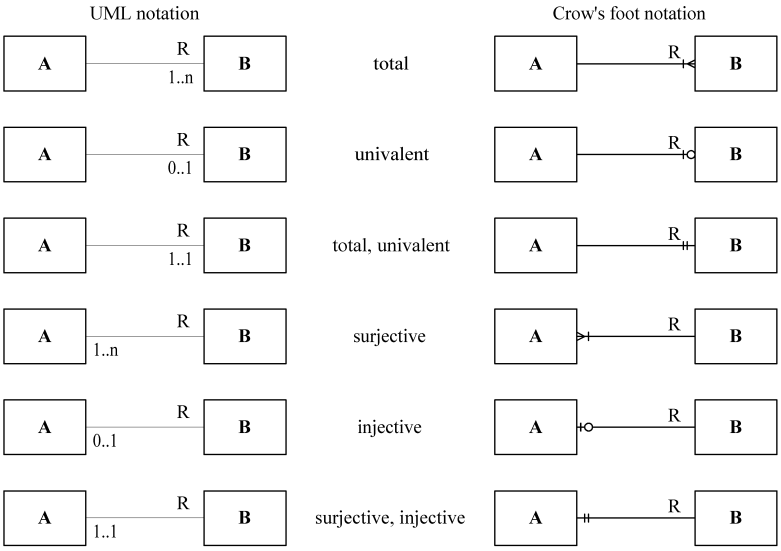

Figure 5: Graphical notations of cardinalities

Some notations are reserved for functions. If $r : A \times B$ is a function, we write $r : A \to B$. In either case we will normally use the symbols f, g, h instead of relational symbols, and the notation $b = f(a)$ instead of $a\ f\ b$. We will

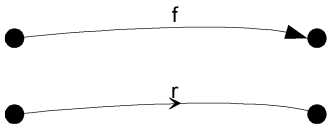

Figure 6: Cardinalities in conceptual diagrams

not denote cardinalities in conceptual diagrams, except for functions. Any relation f that is a function, is depicted by an arrow (figure 6). Every other relation is depicted with the little arrowhead that distinguishes between the left- and right attributes.

References

[1] De Morgan, A. On the syllogism: Iv, and on the logic of relations. *Transactions of the Cambridge Philosophical Society 10*, read in 1860, reprinted in 1966 (1883), 331–358.

[2] Jakobson, I., Booch, G., and Rumbaugh, J. *The unified software development process.* Addison-Wesley, Reading, 1999.

[3] Jipsen, P., Brink, C., and Schmidt, G. *Background Material.* Advances in Computing Science. Springer-Verlag, 1997, ch. 1, pp. 1–21.

[4] Peirce, C. S. Note b: the logic of relatives. In *Studies in Logic by Members of the Johns Hopkins University (Boston)* (1883), C. Peirce, Ed., Little, Brown & Co.

[5] Schmidt, G., Hattensperger, C., and Winter, M. *Heterogeneous Relation Algebra.* Advances in Computing Science. Springer-Verlag, 1997, ch. 3, pp. 39–53.

[6] Schröder, F. W. K. E. Algebra und logik der relative. In *Vorlesungen über die Algebra der Logik (exacte Logik)* (first published in Leipzig, 1895), vol. 3 of *1966*, Chelsea.