7 Relations

7.1 The general idea of a relation

This is a very general concept with many different interpretations. Let's look at the definition:

Relation

If $A_1, A_2, \ldots, A_n$ are sets and R is a subset of $A_1 \times A_2 \times \ldots \times A_n$, then R is called a (n-**ary**) **relation**.

Remember that the elements of $A_1 \times A_2 \times \ldots \times A_n$ are n-tuples $(x_1, x_2, \ldots, x_n)$. Also, note that *any subset* is allowed; there are no conditions for a subset to be called a relation.

Here is an example of a 3-ary or ternary relation. Let A_1 be the set Σ^* of strings, and $A_2 = A_3 = \mathbb{N}$, the set of natural numbers. We consider the relation `staff` $\subseteq \Sigma^* \times \mathbb{N} \times \mathbb{N}$ of all those triples (s, n, m) where s is the name of a lecturer in the School of Computer Science, n is their phone number, and m is their office number. For example, the triple (`Achim Jung`, $44776, 213$) belongs to the relation `staff` but (`Mickey Mouse`, $999, 777$) does not. A picture may be helpful as well:

60

I know of two programming paradigms which are founded on the concept of a general relation: databases and logic programming. In the database language SQL one declares the relation with the command:

61

and then triples can be entered into the relation with the command:

62

In Prolog, the relation does not need to be declared. Individual tuples are entered into it by just stating them as "facts":

63

From the *mathematical* point of view there is not much more to say about general relations, though in database theory we are interested in questions such as whether every element of a set is mentioned in a given relation, or whether for every element in one set there is no more than one element in the other set that is in relation to it.

7.2 Binary relations on a set

In the rest of this chapter we look at the special case of **binary relations** on a **single** set, that is, at subsets of $A \times A$. Apart from being mathematically and computationally interesting, such relations can be interpreted as **directed graphs**, where A is the set of vertices and the relation is the set of edges,[3] and for these we can draw pictures:

64

Binary relations also exist "in nature;" we'll take as a running example the set of all people as A, and the relationship "`likes`" as the relation.

Reflexivity. We call a binary relation $R \subseteq A \times A$ **reflexive** if it contains all pairs (x,x) for x an element of A. As a formula:

Reflexive binary relation.

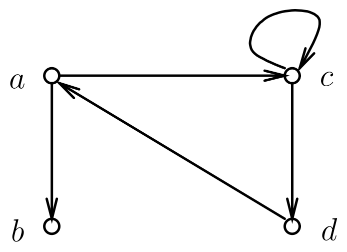
$$R \subseteq A \times A \text{ is reflexive} \overset{\text{def}}{\iff} \forall x \in A.\, (x,x) \in R$$

If we are given a relation R that is not reflexive, then we can make it reflexive by adding the missing edges:

$$\text{refl-closure}(R) \overset{\text{def}}{=} R \cup \{(x,y) \in A \times A \mid x = y\}$$

The new relation refl-closure(R) is called the **reflexive closure** of R. It is the smallest relation that contains R and is reflexive. Note the simplicity of the formula; we don't have to worry whether an edge is added unnecessarily because sets don't carry duplicates in any case. Let's try the reflexive closure on our example relation from above:

65



The opposite concept to reflexivity is that of an **irreflexive relation**, which is one that does not contain any edges leading from a vertex back to itself. The real-world relationship `likes` is neither reflexive (since there are people who hate themselves) nor irreflexive (as there are people who like themselves very much).

Symmetry. A binary relation $R \subseteq A \times A$ is called **symmetric** if with every pair $(x,y) \in R$ we also have $(y,x) \in R$. As a formula:

Symmetric binary relation.

$$R \subseteq A \times A \text{ is symmetric} \overset{\text{def}}{\iff} \forall (x,y) \in A \times A.\, (x,y) \in R \implies (y,x) \in R$$

It is easy to make a relation symmetric; just add the missing links. We develop a bit of notation to write this concisely within the language of sets.

[3] In this interpretation one usually uses the letter V for the set of vertices, and $E \subseteq V \times V$ for the set of edges.

Inverse relation.

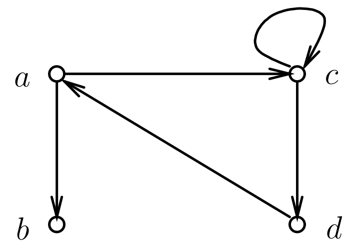
$$R^{-1} \stackrel{\text{def}}{=} \{(x,y) \in A \times A \mid (y,x) \in R\}$$

To make the relation symmetric we add the inverse:

$$\text{symm-closure}(R) \stackrel{\text{def}}{=} R \cup R^{-1}$$

The result is called the **symmetric closure** of R; it is the smallest relation that contains R and is symmetric. Let's apply this to our example relation:

66



As you can see, the drawing of a symmetric graph always has links in both directions between vertices, or none at all.

The opposite concept is that of an **anti-symmetric relation**. Here it is forbidden that any two *different* vertices a and b are connected in both directions. It is allowed, however, for an element to be connected to itself, and also, for two different elements not to have any connection at all.

The real-world relationship `likes` is not symmetric (as there are many instances of person a liking person b but not the other way round) nor is it anti-symmetric (or there wouldn't be any "happily ever afters.").

Transitivity. A binary relation $R \subseteq A \times A$ is called **transitive** if whenever $(x,y) \in R$ and $(y,z) \in R$ then also $(x,z) \in R$. As a formula:

Transitive binary relation.

$$R \subseteq A \times A \text{ is transitive} \stackrel{\text{def}}{\iff} \forall x,y,z \in A.\, (x,y) \in R \text{ and } (y,z) \in R \implies (x,z) \in R$$

Many relations in mathematics are transitive, for example "less than or equal to" on the set of natural numbers. Also, the relationship "is congruent to" on the set of triangles. Most relationships from the real world are not transitive, and `likes` certainly isn't.

To explain how to change a given relation so that it becomes transitive we first introduce how relations are **composed**.

Composition of binary relations.

Given $R, S \subseteq A \times A$ define:

$$R;S \stackrel{\text{def}}{=} \{(x,z) \in A \times A \mid \exists y \in A.\, (x,y) \in R \text{ and } (y,z) \in S\}$$

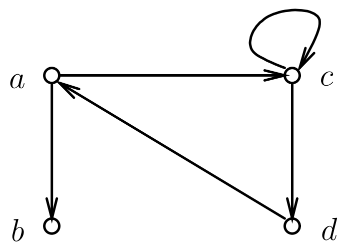
To define the **transitive closure** of a relation R we add all finite compositions of R with itself. As a formula:

$$\text{trans-closure}(R) \stackrel{\text{def}}{=} R \cup R;R \cup R;R;R \cup \ldots$$

The result is the smallest transitive relation that contains R. We note that this is an infinitely long formula. In fact, it can be shown that it is not possible to define transitive closure with the usual means of first-order logic. Since SQL is equivalent to first-order logic, transitive closure is *not definable* in SQL. However, various database vendors have introduced extensions to their systems that allow the user to compute queries such as transitive closure.

In our example we get:

67



The transitive closure operation is related to the idea of a **path**: There is a path from vertex v to vertex v' exactly if (v,v') belongs to the transitive closure of the edge relation.

7.3 Order relations

If we combine the properties introduced in the three items above we get two very important concepts:

Order relation.

A relation $R \subseteq A \times A$ is called an **order relation** if it is reflexive, anti-symmetric, and transitive.

Equivalence relation.

A relation $R \subseteq A \times A$ is called an **equivalence relation** if it is reflexive, symmetric, and transitive.

We take a closer look at each of them.

We first note that the terminology "order relation" is justified, as the relation $\leq$ on the natural numbers (or the integers, or the reals) has the three required properties:

Reflexivity: Indeed, it is true for all numbers x that $x \leq x$.

Anti-symmetry: Yes, if $x \leq y$ and $y \leq x$ then it must be the case that $x = y$. In other words, we cannot have two *different* x and y for which both $x \leq y$ and $y \leq x$ hold.

Transitivity: Again, this is true: $x \leq y$ and $y \leq z$ together imply that $x \leq z$.

The various sets of numbers have another property which is not required for general order relations, namely, for any two numbers x and y we have $x \leq y$ or $y \leq x$. In general, we allow two elements to be **incomparable**, that is, neither (x,y) nor (y,x) belong to the order relation. An example is *divisibility*: Although it is reflexive, anti-symmetric, and transitive, there are incomparable numbers (with respect to divisibility), for example 3 and 5.

An example from computer science is given by *precedence graphs* which indicate which file must be compiled before which other files. "Makefiles" implement this idea.

Small order relations can be drawn as graphs where all edges point upwards and where those edges that follow from transitivity are left out. Here is an example based on divisibility:

68

7.4 Equivalence relations

You can think about an equivalence relation as being a "loose form of equality;" indeed, equality itself is (very trivially) reflexive, symmetric, and transitive. A better example (from mathematics) is the *congruence of natural numbers* which we discussed in Section 2.4. Recall that we say that x and y are *congruent modulo m* if x and y leave the same remainder when divided by m.

For an example more directly from computer science, we can look at ways to compare two implementations P_1 and P_2. Here are some possible equivalences, increasingly liberal:

1. P_1 and P_2 are actually the same file.
2. P_1 and P_2 are in different files, but the files have equal content ("duplicate files").
3. P_1 and P_2 are the same except for formatting ("white space").
4. P_1 and P_2 are the same as before but the order in which variables and methods are declared may now be different.
5. P_1 and P_2 are the same as before but may use different class, variable, and method names ("refactoring").
6. P_1 and P_2 are the same as before but in addition, one may have some methods in-lined ("code optimisation").
7. P_1 and P_2 show exactly the same behaviour when run ("specification" versus "implementation").

(You should know that the School considers every equality of submitted course work, except (7), a case of plagiarism.)

If two elements a and b are related by an equivalence relation, then we say that they are **equivalent**. The symbol that is often used for equivalence relations is $\approx$ (though congruence modulo m is commonly written as $\equiv$ or $\equiv_m$).

For every relation $R \subseteq A \times A$ we can construct the smallest equivalence relation containing it. To do so we simply construct the transitive closure of the symmetric closure of the reflexive closure of R:[4]

$$\text{equiv-closure}(R) \overset{\text{def}}{=\!=} \text{trans-closure}(\text{symm-closure}(\text{refl-closure}(R)))$$

Equivalence relations lead to classifications. If $\approx$ is an equivalence relation on a set A and if a is an element of A, we can look at the subset of all elements that are equivalent to a. This is called the **equivalence class** of a. The notation for this is:

Equivalence class of an element

$$[a]_\approx \overset{\text{def}}{=\!=} \{x \in A \mid x \approx a\}$$

It can be that $[a]_\approx$ is all of A, which means that every element of A is equivalent to a, but more often $[a]_\approx$ will be a proper subset of A. In that case we can look at an element b that does not belong to $[a]_\approx$. It now happens that the equivalence class of b, that is, $[b]_\approx$, is *disjoint* from $[a]_\approx$. The proof of this is very simple: If $[a]_\approx$ and $[b]_\approx$ had an element c in common, then we would have $a \approx c \approx b$ and hence $a \approx b$ by transitivity. This would mean that $b \in [a]_\approx$ contrary to our assumption. A picture may be helpful:

69

[4]The process that produces an order relation from an arbitrary relation is a bit more complex.

On the other hand, if $b \in [a]_\approx$ then in fact $[a]_\approx$ and $[b]_\approx$ must be the same. This is because for every $x \in [a]_\approx$ we have $x \approx a \approx b$ and hence $x \approx b$ by transitivity, and hence $x \in [b]_\approx$ by symmetry.

So we can conclude:

Theorem

If $\approx$ is an equivalence relation on a set A, and a, b are elements of A, then always one of the following two situations is true; either $a \approx b$ and $[a]_\approx = [b]_\approx$, or $a \not\approx b$ and $[a]_\approx \cap [b]_\approx = \emptyset$.

The consequence of this theorem is that an equivalence relation leads to a **classification** of the elements of A, by which we mean a decomposition of A into subsets (usually called “classes”) with the properties:

disjointness No two classes overlap.
coverage Every element of A is contained in some class.

Here is a picture:

70

but I don’t like it so much because it suggests that equivalent elements are somehow close to each other and this could be the wrong intuition. For example, if we take “congruent modulo 3” as our equivalence relation on the natural numbers $\mathbb{N}$, then we get the following picture:

71

We end up with three equivalence classes, which we can characterize globally as: the set of all numbers divisible by 3, the set of all numbers that leave remainder 1 when divided by 3, and the set of all numbers that leave remainder 2 when divided by 3.

Working with classifications via representatives. In a computer it is difficult to work with classifications directly because there is no immediate mechanism for representing sets. Also, the equivalence classes could be infinite sets and then there is no chance at all to represent the whole class. Instead, one works with **representatives** of the classes, that is, with individual elements at the level of the set A, and keeps in mind the notion of equality given by the equivalence relation. A picture:

72

The best example to illustrate this technique is the set $\mathbb{Q}$ of rational numbers which we discussed in Chapter 3. Every rational number can be written as a fraction in infinitely many ways: the fraction $\frac{2}{3}$ is the same rational number as the fraction $\frac{4}{6}$ etc. We can interpret this as saying that the rationals are constructed from the set of fractions (which is just a certain way of writing elements of $\mathbb{Z} \times (\mathbb{Z} \setminus \{0\})$) via the equivalence relation $\approx$ that is defined as

$$\frac{a}{b} \approx \frac{c}{d} \overset{\text{def}}{\iff} ad = bc$$

The arithmetic operations on rational numbers are defined via operations on the fractions:

$$\begin{array}{ccc} \frac{a}{b} + \frac{c}{d} & \overset{\text{def}}{=\!=} & \frac{ad+bc}{bd} \\ \frac{a}{b} \times \frac{c}{d} & \overset{\text{def}}{=\!=} & \frac{ac}{bd} \\ \frac{a}{b} / \frac{c}{d} & \overset{\text{def}}{=\!=} & \frac{ad}{bc} \end{array}$$

This works, because the operations are independent of the chosen representatives. For example, for addition the following is true (and it is a nice exercise in algebra to check this):

$$\frac{a}{b} \approx \frac{a'}{b'}, \ \frac{c}{d} \approx \frac{c'}{d'} \quad \Longrightarrow \quad \frac{ad+bc}{bd} \approx \frac{a'd'+b'c'}{b'd'}$$

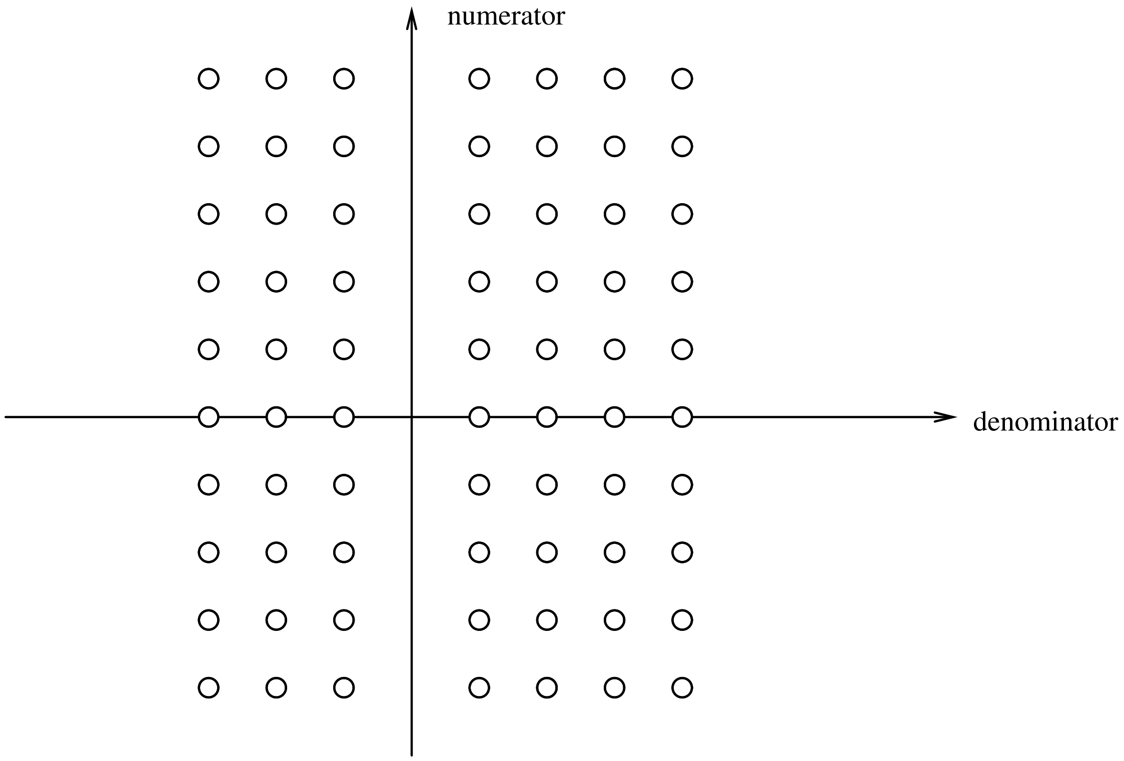
A picture of the situation may also be helpful:

73

An implementation of true rational numbers in Java (or any programming language) follows exactly the same pattern. Rationals are stored as two integers of arbitrary size (using the class `BigInteger`) and the operations are implemented using the equations above. In addition, one would implement the Euclidean algorithm as a method `normalise` for cancelling any common factor in numerator and denominator. When printing a rational, one would first normalise and then print so that the fraction displayed uses numbers as small as possible.

There is a nice 2D illustration of this construction:

74



Exercises

1. Consider the set $V = \{a,b,c,d,e,f\}$ and the relation $E = \{(a,d),(b,c),(c,f),(d,e),(e,a)\}$.
   - (a) Draw a picture of this as a directed graph with vertices V and edges E.
   - (b) Compute (separately) the edge-sets for the reflexive, symmetric, and transitive closure of E (no need to draw the resulting graphs).
   - (c) Compute the edge-set for the equivalence relation generated by E, and draw this as a graph.
   - (d) Write out the classes of the classification which is derived from this equivalence relation.
2. On the set of strings Σ^* over some alphabet Σ consider the binary relation $\lhd$ which holds between two strings s and t if s is a substring of t. Demonstrate that this is an order relation by spelling out what the defining properties mean in this context. Draw the order diagram in the style of Box 68 for all strings over the alphabet $\Sigma = \{a,b\}$ (including the empty string ε) up to length 2.

Practical advice

In the exam I expect you to be able to

- recognize when a given relation is reflexive, irreflexive, symmetric, anti-symmetric, and transitive;
- compute the reflexive, symmetric, or transitive closure of a given relation;
- recognize when a given relation is an order relation, or an equivalence relation (of course, it could be neither);
- draw the graph of an order relation as in Box 68;
- construct the classification that arises from an equivalence relation.