Technische Hogeschool Delft
Onderafdeling der Wiskunde en Informatica

REPRINT OF REPORT 82-11

Set and type denote complementary abstractions

by

J.H. ter Bekke

ABSTRACT

Abstraction can be conceived as a description of the reality in which some details are emphasized while other details are suppressed. Abstractions can be given in two separate ways. First by considering only object identifications and second by considering only object properties.

In data structuring much attention has been given to the use of the set concept from mathematics, and the definition of the concept of type is often based on it. However, this results in descriptions in which only one kind of abstraction appears to full advantage, and the fundamental problems arising from such an approach justify an alternative.

Independent set and type definitions result in operations which are easily formalized and which have repercussions in the use of these concepts in data structuring practice. The concepts denote two complementary abstractions and form an adequate means for describing the reality.

CR Categories and Subject Descriptors:
D.3.3 [Programming Languages]: Language Constructs – Abstract Data Types;
H.2.1 [Database Management]: Logical Design – Data Models.

General Terms: Design, Languages, Theory.

Additional Key Words and Phrases:
data abstraction, relational database, relational algebra, invariants, set theory, set algebra, theory of types, algebra of types, data semantics.

1. INTRODUCTION

The preciseness of mathematics has often significantly contributed to the developments of computer science, but unfortunately, some fundamental mathematical concepts have been introduced incorrectly in this new application area. Substantial concessions were sometimes needed to model the real situation within the limits of an existing mathematical framework with the consequence that essential properties of data were ignored. Conspicuous examples of the consequences of such concessions occur frequently in the use of the set concept.

The concept of set appears directly applicable to data description, since it denotes an abstraction which can be described as “a collection into a whole of definite, distinct objects of our intuition or of our thought” [7]. That an uncritical introduction of this concept in a data processing environment may lead to serious misconceptions can be illustrated through the following examples.

The developments of data structuring have been rather tempestuous during the past decade, mainly due to the introduction of the relational data model [5] in 1970. This model depends heavily on the use of sets and relations from mathematics. In spite of its merits, this approach to data description in a database environment is open to criticism on the grounds of fundamental difficulties arising from it.

In the relational model data re conceived as elements (or tuples) of a set (or relation). For instance, data about employees in an organization are conceived as a set in which data about an individual must be an element of this set. The continuously changing membership of the collection of employees has to be ignored to fit in with the model. This changing membership is an essential property of the reality; in this example changes occur when employees are hires and fired, and this implies that the definite collection of objects for the life span of a database, by definition an invariant in the relational model, is falsified by the reality. In a later stage in the design, the real invariant properties of the reality (i.e. the relation attributes) are recognized because of their key role in normalization procedures [6]. But even then, invariant properties are changed without awareness of their consequences in the application area of the data model [4].

These shortcomings of the relational model can not be bypassed by solutions based on more complicated set concepts such as power sets. All efforts in such a direction are doomed to failure because of the inherent inadequacy of the set concept in this context. Application of the set concept in the database area emphasizes transient properties of the reality and suppresses invariant (and therefore essential) properties of the reality. This contention will be elaborated in the present paper.

Similar efforts to describe data using the set concept from mathematics are found in the programming language area. Gries presents an overview of these in [8]. From this overview it appears that an adequate definition of the concept of (data) type has been sought for a long time in this area. One of the most promising approaches is the inclusion of basic operations in the definition of type.

The approaches in programming languages are similar to the approaches in databases; a type is conceived as a set with some additional properties. But data structures are not determined by a definite collection of elements, indeed data processing would be impossible if they were, which should imply that the set concept cannot be used for their description.

The set concept approach is not used by Liskov and Zilles [11], who recognize that a type is completely characterized by certain basic operations. This implies an abandonment of type instances. However, the paper is concerned with programming languages and their approach cannot easily be transferred to provide a generalized approach to databases. Such an approach is attempted in the present paper.

None of the previous approaches has found general acceptance, but one is needed for further developments of computer science. This emerges from a recent workshop for researchers in the areas of artificial intelligence, programming languages and databases [15] where a complete session was devoted to the search for a generally acceptable approach.

The search for more fruitful new concepts must start with a careful analysis of already existing concepts. An analysis of the mathematical set concept is therefore presented in section 2. This is followed by a reformulation of the set concept in section 3. An examination of specific properties of data processing leads to a definition of the concept of type in section 4. The concept of type is complementary to the earlier defined set concept. Operations on types are therefore trivial, and are close to abstractions which are considered in linguistics and philosophy.

In section 5 some advantages of the new approach are presented. These include comparisons with similar approached in the database area. Disadvantages of the relational algebra [5] as a basis for database design are emphasized. A comparison with database abstractions [12, 13] indicates that a clearer distinction between the concepts of set and type requires reformulation of the notions of aggregation and generalization.

It should be stated that critical remarks on earlier developments in databases and programming languages are made in full awareness that most of them were indispensable contributions to the development of our field. This applies especially to the seminal publications by Codd [5], Smith and Smith [12, 13], Hoare [10], Liskov and Zilles [11]. Finally the overview by Gries was indispensable for the present paper.

2. ABSTRACTION

Reality is so diverse that the only way to be able to interact with it is to suppress the majority of details so that a manageable collection of details remains. The resulting description is called an abstraction. Abstraction is concerned with data and is therefore a prime research domain for data processing and especially for databases.

Abstractions can be given in two separate ways: aggregation and classification. Aggregation is an abstraction in which a collection of object identifications is conceived as a single identification of a new object. Object properties are suppressed in this abstraction. Classification is a complementary abstraction of the reality. Object properties are collected while object identifications are suppressed. The choice of abstraction depends mainly on the manageability of the collected details.

A simultaneous consideration of object identifications and object properties would result in a complete description (i.e. copy) of the reality and therefore could not be called an abstraction, but such an effort would fail because of the unmanageable mass of details.

Any model of (a part of) the reality consists of details from the reality which are assumed to be invariant for (that part of) the reality. Transient details cannot be considered as part of any model. Application of this principle on data structuring would imply that a data model for an organization should consist of only those objects which are invariant for that organization. Invariant objects could be: employee, department, project, product, supply and order. Individual objects, such as employee X, cannot be handled by any model because they denote transient properties of the organization. It would therefore be meaningless to define the collected objects by means of individuals. This implies that the invariant objects cannot be conceived as aggregations, but instead as classifications.

Descriptions of the reality are given using assertions [2, 3]] which consist of subject and (defining) predicate. These concepts are essential for the abstractions that can be distinguished. The necessity of such a conception can be illustrated as follows.

Suppose that the collection {A, B, C} is given. This collection is defined by enumerating the elements identified by A, B and C. Element identifications are needed for the description of the predicate {A, B, C}. Naming elements however implies naming sets and therefore naming subjects. We could name this set by S = {A, B, C} or alternatively by:

    <u>set</u> S = A, B, C

which is more convenient for the subsequent discussion.

The definition above is conceived as a assertion in which the left part denotes the subject (i.e. the new set S) and the right part denotes the predicate (i.e. the existing sets A, B, C). This example illustrates the necessity of assertion as structuring concept.

Let us now consider the set G = {A, B, {D, E}}. The element definitions are now denoted as follows:

set A = ...
set B = ...
set ...= D, E

In these definitions … denotes unknown parts. From these definitions it follows that elements are given as a mixture of subjects and predicates. Such an approach leads to unacceptable misinterpretations and undecidability, as illustrated below.

Because a set is defined as a definite collection of distinct elements, "it should be intrinsically settled for any possible object C whether C is a member of the set or not" [7]. In our example this proposition would only be decidable if C is completely defined in terms of subject and predicate. But such an approach would lead to a complete description of the reality.

To prevent misinterpretations and undecidability we require the use of either subjects or predicates in a single defining predicate. So, defining predicates of the form {{X}, {Y}, {Z}} are still allowed. Each element is now described by a predicate. The relevant question is now: can a collection of predicates be conceived as a set?

In case of a set we could write T = {{X}, {Y}, {Z}}. This, however, leads directly to unacceptable situations in which both subjects and predicates are considered as sets. So the only acceptable solution which remains is to conceive a collection of predicates as a predicate, hence {T} = {{X}, {Y}, {Z}}. We call this a type definition and denote this abstraction as follows:

type T = X, Y, Z

By this discussion we arrive at two disjoint classes: set and type. Operations are only defined within a single class. For sets this includes operations such as union, intersection and difference. For types, operations like specialization, generalization and differentiation are introduced.

It is now clear why sets are defined by object identifications (subjects) and types by object properties (predicates). In our previous example employee is conceived as a type which is defined by the properties: name, address, city, department and salary. The set of employees is called personnel and is defined by the totality of individuals in the organization (i.e. the employee identifications). This is also illustrated in figure 1.

This informal introduction to the concept of set and type will be followed by a more formal treatment of these concepts in subsequent sections of this paper.

3. SETS

Set is a fundamental mathematical concept. This concept will receive special attention in this paper to show it differs from the concept of type. Its definition will be followed by some examples to illustrate its use in practical situations.

The set concept denotes a form of abstraction which is defined as follows:

> A set is a collection into a whole of definite, distinct object identifications. An object with an identification of the set is called an element of the set.

The following characteristics are derived from the definition above:

- a set is a collection into a whole; the creation of a set is therefore a form of abstraction which is known as aggregation.
- a set consists of a definite collection of object identifications; a set has therefore a definite collection of corresponding elements.
- a set consists of a collection of distinct object identifications; this means that the elements are different.
- object properties are irrelevant for a set; this expresses the variable part of the elements.
- the collating sequence is irrelevant for a set.

Most remarkable in the definition above is the fact that a set is completely characterized by a collection of object identifications. This is called the invariant property of a set. Object properties are irrelevant as illustrated in the following examples.

1. Consider the symbols a, b, c, …, z. The set which consists of these 26 symbols is called an alphabet.
2. Consider the objects chassis, coach-work, motor. The set which corresponds with this collection is called a car.
3. Consider pitcher, catcher, first base, …, center field. The corresponding set is called a baseball team.
4. Consider all individuals who are employed for an organization. The corresponding set is called personnel.
5. In mathematics we may define the following sets:
   - the set of prime numbers between 2 and 6;
   - the set of odd numbers between 2 and 6.

   The descriptions above denote the same set consisting of the elements 3 and 5. This illustrates that object properties are irrelevant for the set.
6. The concept of assertion may be conceived as a set in which subject and predicate are the two elements.

Operations are also needed for sets. The following collection of operations is called the set algebra. Their definitions require the use of the invariant property of the set.
The relevant operations for sets are:

- the <u>union</u> of the sets A and B is the set consisting of identifications which occur in A or occur in B;
- the <u>intersection</u> of the sets A and B is the set consisting of identifications which occur in A and occur in B;
- the <u>difference</u> of the sets A and B is the set consisting of those identifications in A which do not occur in B.

These operations are intuitively illustrated by Venn diagrams, in which circles denote sets. The results of set operations are shaded in figure 2.

The empty set is defined in mathematics as the unique set without any element (i.e. object identification). This set results, for instance, from the intersection of two disjoint sets.

The universe is sometimes introduced together with the empty set as the set which contains "everything". As is well-known, such an approach results in antinomies [7]. We consider the universe in the next section.

4. TYPES

Though the concept of type is important for data processing, it is nevertheless often inappropriately defined, with irrelevant aspects being emphasized, and relevant aspects neglected. This can be illustrated as follows. Suppose that data must be described about employees in an organization. A structure is then perceived in which data about an individual are considered as an element in a set. In such a structure properties are subordinate. That properties must play an important role in the design of a data model is observed later on when the data structures are normalized [6].

We need a concept for data description which allows us to emphasize the object properties, the invariant part of the data. Based on this principle we define the concept of type as follows:

> A type is a collection into a whole of definite, distinct object properties. An object with the properties of the type is called an instance of the type.

The following characteristics are derived from the definition above:
– a type is a collection into a whole; the creation of a type is therefore a form of abstraction which is known as classification;
– a type consists of a definite collection of object properties; a type has therefore an irrelevant number of instances;
– a type consists of a collection of distinct object properties; this means that instances differ through their property values;
– object identifications are irrelevant for a type; this expresses the variable part of the instances;
– the collating sequence is irrelevant for a type.

Most remarkable in the definition above is the fact that a type is completely characterized by a collection of object properties. This is called the invariant property of a type. Object identifications are irrelevant as illustrated in the following examples:

1. Consider the symbols a, b, c, …, z. The corresponding type is called letter. This type could be defined as in a dictionary with the property “represents a speech sound” of briefly “sound”.
2. Consider the properties: name, position, club. The corresponding type could be called player.
3. Consider the properties: name, address, city, department, salary. The corresponding type is called employee.
4. The concept of set as defined in the previous section denotes a type. It could be defined with the properties: abstraction, object identification.
5. Operations may also be conceived as object properties. Fundamental mathematical concepts such as group, ring and field are obtained by specializations from the concept of set.
6. The concepts of set and type may be generalized into the concept of abstraction. This is defined by the property “collection into a whole” or briefly “membership”. Semantic abstractions however are obtained by using its specializations set and type.

The immediately preceding examples have illustrated the use of types in various situations. People learn this form of abstraction at an early stage as well as the use of operations on types. Examples might be:

generalization (e.g. animal is a generalization of dog, cat, …)
specialization (e.g. the properties of cat enclose the properties of animal)
differentiation (e.g. there exist animals other than cats).

We will now introduce these operations in a more formal way. The collection of operations for types is called the algebra of types. The definitions require use of the invariant property of the type, and instances play no part. The relevant operations for types are:

- The specialization of the types A and B is the type consisting of properties which occur in A or occur in B.
- The generalization of the types A and B is the type consisting of properties which occur in A and occur in B.
- The differentiation of the types A and B is the type consisting of those properties in A which do not occur in B.

These operations are intuitively illustrated using Venn diagrams. Here, circles denote objects with the properties a, b or c (e.g. a: is road vehicle, B: is vessel, c: is motorized). Instances of the types A and B having the properties a,c and b,c respectively are given in figure 3. Instances of their specialization, generalization and differentiation are given in figure 4.

The previous section contains the definition of the empty set. In this section we can define its complement using the concept of type. Reduction of properties denotes generalization. It is therefore obvious to define the universe as being the unique type without properties. This means that the universe contains all possible observations without corresponding data (i.e. properties). This concept relates to the propositions that reality (or universe) is fact does not mean anything.

5. COMPARISONS

The previous section contains a type definition in which instances do not play any role. The resulting concept of type for data structuring is analogous to the concept of repetition in programming, in that both concepts originated from a contemplation of particular instances. In both cases deeper insights have resulted in complementary views.

The developments are easily traced in the programming area. Originally flowcharts were used for program documentation. Flowcharts, however, suggest a contemplation based on instances or test cases. Programs which originate from such an approach are often unreliable. Currently, repetition instances are considered as subordinate to the concept of invariant in the programming area (see also Gries [9]).

In data structuring similar developments are to be expected. Originally data structure diagrams were used for documentation. These diagrams are similar to flowcharts. They encourage a thought process in which examination of instances predominate (compare this with the so-called one record at a time logic).

The relational data model may be considered as the first step in a direction in which instances are subordinate. This model stimulates an analysis of invariant data object, but it contains some operations which originate from a contemplation of instances. This will be explained in the following informal discussion in which relational algebra operations are analyzed.

The cartesian product is an operation which enables us to create a new set from existing sets. The elements from the resulting set are completely determined by the existing sets. Such an approach can lead to strange situations as illustrated in the following example about students and courses. The cartesian product of these sets could be called enrollments. However, it does not follow that all possible (student, course)-combinations occur in the resulting set. In the relational model we use therefore relation instead of cartesian product, hereby denoting a subset of a cartesian product. But this subset is not determined by a definite collection of elements (i.e. tuples). This collection is only determined by a definite collection of relevant properties. This indicates that the concepts from the relational model emphasize irrelevant details, while relevant aspects are almost forgotten. It is therefore advisable to drop sets and cartesian products from data models because they can easily lead to wrong conclusions.

An alternative conception in terms of types emphasizes the essential characteristics of the desired abstraction. Student and course types may be used as a starting point for the definition of the enrollment type. We then have considered only invariant properties of types. Instances are not discussed at all. Observations of the reality can provide relevant instances of these types.

The problems mentioned above are also encountered when other operations of the relational algebra are used. Join and projection are typical examples of operations which may easily lead to misinterpretations. This can be illustrated as follows.

Join is an operation which results in a relation having more attributes than the existing relations (i.e. operands). An increase in the number of attributes results also from specialization. Specialization however can never lead to an increasing number of instances, because every instance from the specialized type occurs also as an instance from the existing types (i.e. operands).

Joins may result in larger sets. For example the join of the two relations P and Q (see figure 5) results in a relation R which contains more tuples than the original relations P and Q.

The situation above is similar to the discussion about cartesian products. It would be advisable to consider subsets of joins (which may possibly be empty). But also in this case it is impossible to denote a definite collection of elements. In other word, by using the concept of join we have emphasized irrelevant details of data. A contemplation in terms of types considers only relevant details of data. The problem termed “connection trap” [5] cannot occur in a model based on types.

The discussion above should also lead to a rejection of relational algebra and relational calculus as candidates for data manipulation. An alternative to these languages is presented in [1].

Smith and Smith have drawn attention to an approach which is known as database abstractions [12, 13]. They require each object to be named with a meaningful noun. For instance: the generalization of a number of object $O_1, \ldots, O_n$ may only be defined as object O, if all objects are named by a noun [14]. Such an approach requires that the semantics of object types be completely determined by type names.

In the present paper we have reformulated the concept of generalization so that the semantics are determined by the defining properties. This does not imply that meaningful names cannot simplify this process [3]. Such an approach will be illustrated with an example that has been derived from [13]. The fact that car and airplane may be generalized into vehicle is not determined by the type names car and airplane, but by the type properties. Hence the generalization of:

type car = manufacturer, price, weight, medium, horsepower
type airplane = manufacturer, price, weight, medium, maximum altitude

consists of the type:

type vehicle = manufacturer, price, weight, medium

Optimal use of the coherence between these type definitions would result in a vehicle definition as above and the other type definitions as below:

type car = vehicle, horsepower
type airplane = vehicle, maximum altitude

This implies the validity of so-called invariants e.g. the subset invariant [1] and convertibility [2]. The subset invariant denotes for example the fact that a car is a vehicle. Convertibility denotes for example that a car is defined by the properties vehicle and horsepower and nothing else. It is obvious that these invariants do not have to be stated explicitly because they form an intrinsic part of the definitions.

6. CONCLUSIONS

A description of the reality is given using assertions. An assertion consist of two sets: object identification (i.e. subject) and object properties (i.e. defining predicate). This implies that two types of abstraction can be considered: aggregation and classification. Aggregation emphasizes only object identifications, while object properties are suppressed. This form of abstraction corresponds with the familiar set concept. Classification emphasizes only object properties, while object identifications are suppressed. This form of abstraction corresponds with the type concept. Their complementary character is also illustrated in figure 6.

Both abstractions suppress details of the reality. Because they consider complementary details, they form an adequate means for describing the reality. Other abstractions cannot exist because a description in terms of both object identifications and object properties is only suitable for a finite and static reality.

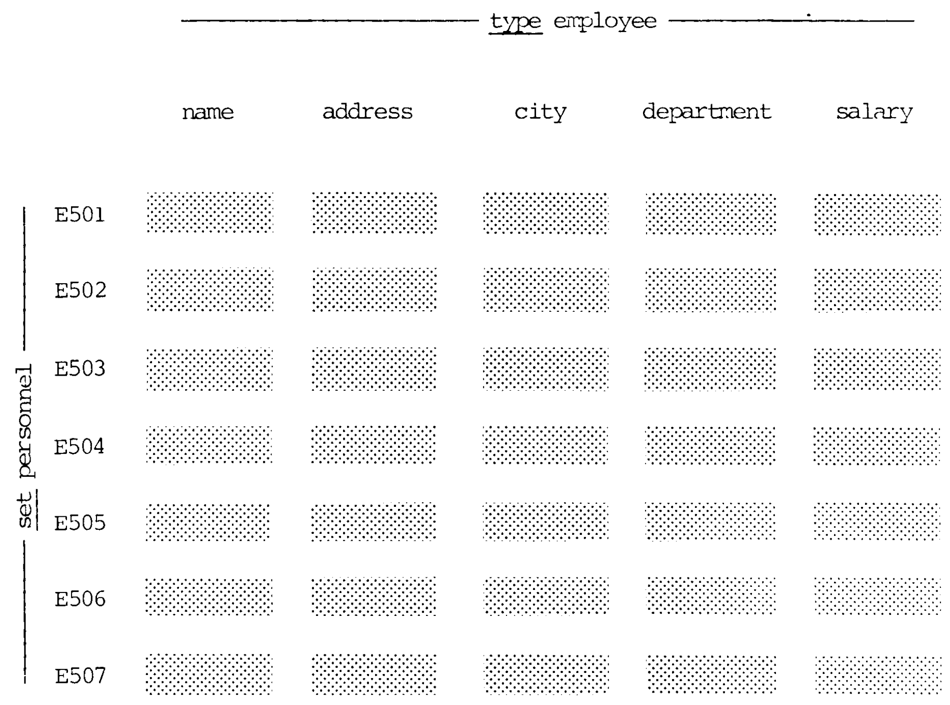
A data model consists of invariant details of a certain part of the reality. In pure mathematics the nature of objects is always suppressed, hence the suitability of the set concept in this context. However in data processing we have to deal with a complementary situation. Elements may vary, but nor their properties. This implies that the set concept is no longer a useful description of the reality in a data processing context. Considering only invariant properties of data results in an alternative in which the concept of type is dominant. This approach is considered as a strong candidate for data modeling because it consists of basic structures (including semantic invariants) and operations on these structures.

ACKNOWLEDGMENT

The author is indebted to his colleagues A. van der Ende and D.J. McConalogue for a careful reading of the text and helpful suggestions for improvements, most of which have been incorporated.

REFERENCES

1. J.H. ter Bekke, A Data Manipulation Language for Relational Data Structures, Systems for Large Data Bases (ed. P.C. Lockemann, E.J. Neuhold), North-Holland, Amsterdam, 1976, pp. 159-168.
2. J.H. ter Bekke, Convertibility in Databases, Information Processing Letters 11 (4, 5), December 1980, pp. 168-171.
3. J.H. ter Bekke, Equilibrium in Databases, (submitted for publication).
4. P.A. Bernstein, N. Goodman, What can Boyce-Codd Normal Form do?, Proceedings 6th International Conference on Very Large Data Bases (1980), pp. 245-269.
5. E.F. Codd, A Relational Model of Data for Large Shared Data banks, Comm. ACM 13 (6), June 1970, pp. 377-387.
6. E.F. Codd, Further Normalization of the Data Base Relational Model, Data Base Systems, Courant Computer Science Symposia Series, Vol. 6, (ed. R. Rustin) Prentice-Hall, Englewood Cliffs (1972), pp. 33-64.
7. A.A. Fraenkel, Abstract Set Theory, North-Holland, Amsterdam (1976).
8. D. Gries, Introduction to Part IV: Data Types, Programming Methodology (ed. D. Gries), Springer-Verlag, New York, 1978, pp. 263-368.
9. D. Gries, Educating the Programmer: Notation Proofs and the Development of Programs, Information Processing 80 (ed. S.H. Lavington), North-Holland, Amsterdam, 1980, pp. 935-944.
10. C.A.R. Hoare, Notes on Data Structuring, APIC Studies in Data Processing No. 8: Structured Programming, Academic Press, London, 1972, pp. 83-174.
11. B. Liskov, S.N. Zilles, Programming with Abstract Data Types, ACM SIGPLAN Notices, April 1974, pp. 50-59.
12. J.M. Smith, D.C.P. Smith, Data Base Abstractions: Aggregation, Comm. ACM 20 (6), June 1977, pp. 405-413.
13. J.M. Smith, D.C.P. Smith, Data Base Abstractions: Aggregation and Generalization, ACM TODS, June 1977, pp. 105-133.
14. J.M. Smith, D.C.P. Smith, Principles of Database Conceptual Design, Proc. NYU Symposium on Database Design, New York, May 1978, pp. 35-49.
15. Proceedings of the Workshop on Data Abstraction, Database and Conceptual Modeling (ed. M.L. Brodie, S.N. Zilles), SIGART Newsletter 74 (Jan. 1981), SIGMOD Record 11, 2 (Febr. 1981), STIGPLAN Notices 16, 1 (Jan. 1981), pp. 43-52.

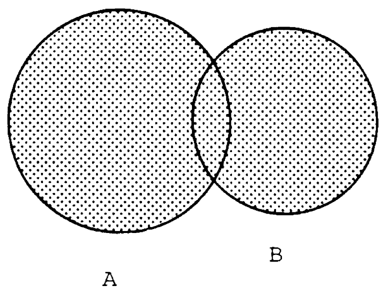


<u>type</u> employee = name, address, city, department, salary

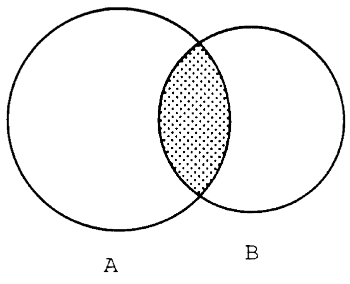
<u>set</u> personnel = E501, E502, E503, E504, E505, E506, E507

<u>Figure 1: Illustration of the set and type concepts</u>

Union



Intersection



Difference

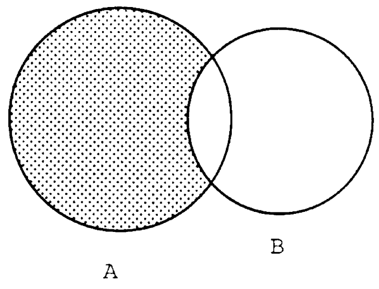
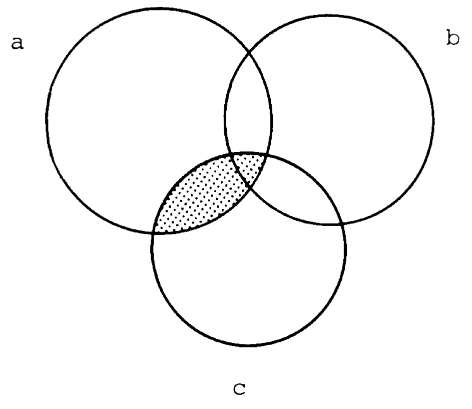


Figure 2 : Set operations

Type A



Type B

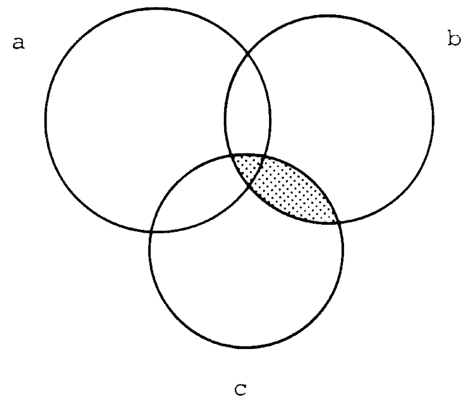
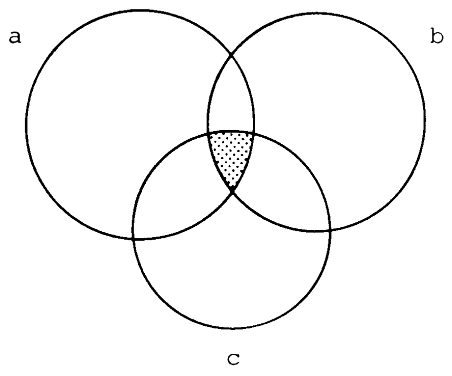
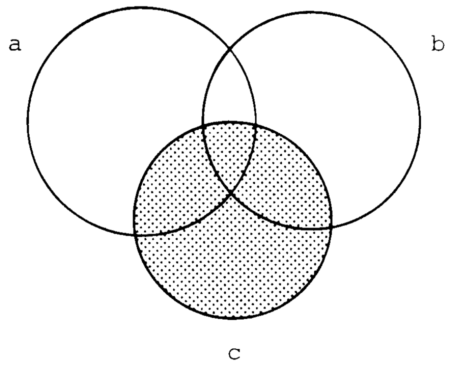


Figure 3 : Instances of types A and B

Specialization



Generalization



Differentiation

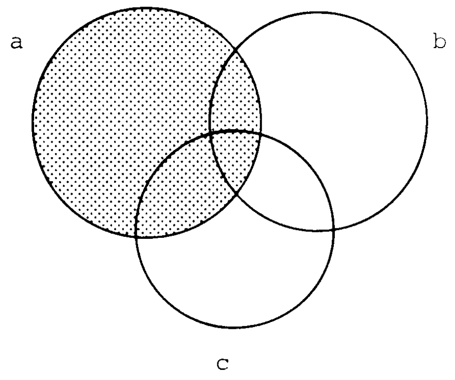


Figure 4 : Type operations

P(a	b)
1	2
2	2
3	1

Q(b	c	d)
2	1	2
2	3	4
2	2	0

R(a	b	c	d)
1	2	1	2
1	2	3	4
1	2	2	0
2	2	1	2
2	2	3	4
2	2	2	0

Figure 5 : Join of two relations

	subject	predicate
set	relevant	irrelevant
type	irrelevant	relevant

Figure 6: Abstractions