Report Number: WUCS-82-10

1982-08-01

Semantic Abstraction and the Concept of Type

Takayuki D. Kimura

SEMANTIC ABSTRACTION
AND
THE CONCEPT OF TYPE

Takayuki Kimura

WUCS-82-10

August, 1982

(Revised May, 1983)

Department of Computer Science
Washington University
St. Louis, MO. 63130

1. Introduction

The concept of type plays an important role in data modelling because it is a mechanism for preserving the data integrity and semantic consistency of a database. The purpose of this report is to explain the philosophy behind the concept of type in a formal data model called Abstract Database System (ADS). Our philosophy is based on the Russell's definition [6]: "A type is defined as the range of significance of a propositional function, i.e., as the collection of arguments for which the said function has values. The theory of type is a theory of symbols rather than a theory of things." (Note that Russell's "range" corresponds to "domain" in modern mathematics.) We will not describe in detail the syntax and semantics of the ADS constructs for type specification and type checking. They can be found in [2,3,5]. However, we will present enough of the syntax and semantics to make this report self-contained.

As a part of a research project to develop a design methodology for medical information systems, ADS has been proposed as the nucleus of a database design methodology where data abstraction and data integrity are essential properties of a database management system. The ADS mechanism for data abstraction is the abstraction (lambda) operator of the lambda calculus [1]. A lambda expression in ADS is called a descriptor and can be used to represent both a function and a set. The mechanism for data integrity consists of two parts: consistency checking between generalization and instantiation, and consistency checking between intensional data and extensional data. The concept of type is used for the first part. The concept of logical implication is used for the second part. We will not deal with the second part in this report.

An abstraction variable in an ADS descriptor (lambda expression) is a name that follows the lambda symbol. A type name which is associated with a variable name specifies the symbols that can be substituted for the variable in the abstraction body. The type specifies the domain of abstraction (generalization) represented by the lambda expression. If the generalization is instantiated by a symbol that does not satisfy the type specification, the resulting expression will be considered non-significant, i.e., it may be syntactically correct but has no meaning. The type checking capability of ADS guarantees that the data representing generalizations are always consistent with the data representing their instantiations.

Since a type is defined as a collection of symbols and a set can be represented by a lambda expression, a type specification in ADS is a special form of the abstraction on symbols. Therefore, including type checking in ADS did not increase its complexity. Any data model which has the abstraction capability can easily handle type checking.

In the next section, we will introduce the semantic concepts and notation necessary for later discussion. In section 3, we will define the concepts of syntactic and semantic abstraction. In section 4, we introduce the concept of type. In section 5, we present conclusions.

2. Preliminaries

In this section, we will introduce different semantic relations based on Frege's sense/denotation dichotomy and discuss the concept of nonsignificant symbol.

2.1 The Meaning of Meaning

According to Frege [4], a symbol always has two different meanings; sense and denotation. In other words, there are different meanings associated with the word "meaning". For example, suppose that Jack is Kathy's brother. Then, consider the following two sentences:

(1) "I know Jack"

(2) "I know Kathy's brother".

If the two symbols "Jack" and "Kathy's brother" have the same meaning, then whenever (1) is true so is (2). However, a situation can exist in which (1) is true but (2) is not true (e.g., when "Kathy's brother" refers to to one of Kathy's other brothers). Therefore, we must conclude that the two symbols have different meanings in spite of the fact that in this case they refer to the same person. We say that they denote the same denotation, Jack as a person, and express different senses. Two symbols are equivalent (=) if they have the same denotation. They are synonymous (:=:) if the two symbols have the same sense. For example,

"Jack" = "Kathy's brother"

not ("Jack" :=: "Kathy's brother").

Note that synonymity implies equivalence, i.e., for arbitrary symbols A and B, if A :=: B then A = B.

When a symbol is a sentence consisting of simpler symbols (words), its denotation is a truth-value, Truth (T) or Falsehood (F), and its sense is a proposition expressed by the sentence. Consider the following sentence:

(3) "Jack is a man".

With the understanding that Jack is Kathy's brother, (3) denotes T and

expresses a proposition that a person called "Jack" is a male human being. Note that a proposition is not a symbol but an abstract object.

Since any symbol may represent itself, we will use the convention of quoting a symbol to refer to the symbol itself, and the convention of surrounding a symbol by a pair of square brackets to refer to the symbol's denotation. In summary, a symbol such as (3) may have three different meanings that can be referred to as:

sentence:	"Jack is a man"	
proposition:	Jack is a man	
denotation:	[Jack is a man]	(= Truth).

2.2 Nonsignificant Symbols

There are sentences that are neither true nor false, but rather nonsignificant. They are grammatically correct but express nonsense. The following are examples of nonsignificant sentences:

"2+3i < 1-2i", where i = -1.

""Boston" is a large city."

"Boston consists of six characters."

"This sentence is false."

There may be some disagreement among native speakers as to whether the above sentences except for the last one are nonsignificant or simply false. It depends on the granularity of semantic analysis that is accepted by native speakers. The last sentence, however, will lead to a well known paradox if we assume that it is either true or false. Therefore we must conclude that the last sentence is nonsignificant.

Note that the following sentences are no longer nonsignificant, but are simply false:

""This sentence is false" is true."

""This sentence is false" is false."

In general, for an arbitrary symbol x, the following sentences are always significant and are either true or false, even if 'x' may not be significant:

''x' is true'

''x' is false'

''x' is significant'

''x' is nonsignificant'.

where the single quotes indicate that the quoted symbol has a free occurrence of a variable into which a symbol can be substituted to produce a new symbol. It represents a generalized form of a quoted symbol. The usage of the single quotes here is the same as in ADS.

The ADS notation for testing the significance of a symbol is:

'/x/' :==: 'x is significant'

where x is an arbitrary symbol and ":==:" means "is synonymous to by definition". We will also use the following convention:

'x :=: Ø' :==: 'x is nonsignificant'.

For example, "This sentence is false" :=: Ø,

/"This sentence is false"/ = F.

3. Semantic Abstraction

Many other researchers have pointed to the importance of data abstraction in database management and software system management [7]. ADS offers abstraction on symbols as a basic tool for information management. There is a major difference between our notion of abstraction and theirs. Our view is that abstraction is possible only through a symbolism. Their view is that abstraction is independent of symbolism. Our theory of abstraction is a theory of symbols. When a

user represents knowledge in a sentential form, he has abstracted reality at the first level. This kind of abstraction, however, is beyond the scope of our formal study.

There are two complementary mechanisms of abstraction in ADS, generalization and discrimination. Generalization is an abstraction of similar patterns (symbols) into one general pattern (symbol) which identifies the similarity. Discrimination is an abstraction of similar patterns into one discriminating pattern by which differences can be identified. We will discuss only generalization because an ADS type is a generalization of significant sentences.

Now consider two sentences:

(3) "Jack is a man"

(4) "Dave is a man".

The sentences are similar as a symbol and express similar propositions. Assuming that Dave is also Kathy's brother, the two sentences also have a similar (in fact, the same) denotation, Truth. Based on these similarities, we can generalize (3) and (4) at three different levels: (a) what it is as a symbol, (b) what it expresses (its sense), and (c) what it denotes. The generalization at the third level (c) is well understood and the simplest, i.e., the result of generalization is the truth-value Truth as follows:

"Jack is a man" = "Dave is a man" = Truth.

We will separately investigate generalizations at levels (a) and (b) below.

3.1 Syntactic Abstraction

By syntactic abstraction we mean generalization on symbols at the level (a). Considering (3) and (4) as two complex symbols, we can

identify "is a man" as the common occurring pattern, and names of people as the differentiating part. Thus, we observe that they are two instances of a more general sentence form:

'_____ is a man'

where the blank _____ can be filled with any person's name.

In ADS, the above generalization of sentences can be named and can be specified as follows:

(5) Man == (λx:person-name)'x is a man'

The symbol "λ" is called the abstraction (lambda) operator, "x" is the operator variable, "person-name" is the type of the variable (we say "x is of type person-name"), and " 'x is a man' " is the operand.

The single-quotes indicate that the abstraction is on the first level, i.e., on symbols. The double-quotes are not used because the operand 'x is a man' is not a symbol by itself. The operand becomes a symbol only when some symbol of type person-name is substituted for the operator variable "x" in the operand. A single-quoted symbol denotes the double-quoted symbol only after some symbol has been substituted for all occurrences of variables within the single-quotes. A double-quoted symbol denotes the symbol itself and no substitution is allowed. A double-quoted symbol is a constant with a definite denotation (the symbol itself), and a single-quoted symbol is a variable without such a denotation.

The relationship between the individual sentence (3) and the general sentence form, Man, can be designated in ADS as follows:

"Jack".Man :=: "Jack is a man"

The lefthand side refers to the symbol denoted by the operand of Man after the symbol "Jack" is substituted for all occurrences of the

operator variable of Man, "x", in the operand. Note that

"Unicorn".Man :=: Ø

Jack.Man :=: Ø

because "Unicorn" is (presumably) not a person's name, and Jack is not a name but a person. In other words, the expression ""Unicorn"" in ""Unicorn".Man" is not of the type person-name because it does not denote a person's name. Similarly, the expression "Jack" in "Jack.Man" is not of the type person-name because it does not denote a person's name. Instead, "Jack" is of type person because it can be inferred from the fact that "Jack is a person" is true.

Note that this particular generalization implies that sentences such as "Unicorn is a man" are not false, but, instead they are nonsignificant (or meaningless). They do not express any sense (proposition), and therefore denote nothing. By the same token, "Jack.Man" is nonsignificant and expresses no proposition; while "Kathy is a man" is significant and denotes the truth-value F.

There are two ways of defining the denotation of the name "Man", or the symbol "(λx:person-name)'x is a man'":

(a) It denotes a class of sentences (symbols) sharing some common properties.

(b) It denotes the function which operates on a symbol of the type person-name as its argument, and yields a sentence as its value.

In the first case, ""Jack".Man" identifies a particular member of the class, and in the second case, the symbol "." denotes the application of a function to an argument, i.e., ""Jack".Man" is equivalent to "Man("Jack")" in mathematical notation. The domain of a function is the same as the range of the operator variable. The value of a function is

undefined if an argument has an incorrect variable type. It is a partial function from symbols to symbols.

3.2 Semantic Abstraction

By semantic abstraction, we mean a generalization on propositions at level (b). The proposition expressed by (3) relates, rightly or wrongly, the person called Jack to the property (or attribute) of male humanity. Similarly, the proposition expressed by (4) relates the person called David to the property of male humanity. The similarity between the two propositions can be characterized by the relationship between the subject, a person, and the property of male humanity. It follows that the two propositions can be generalized into a more general proposition.

In ADS, this general proposition can be named and specified as follows:

(6) man == ()x:person)(x is a man).

Note that "man" is the name of a generalized proposition, while "Man" in (5) is a name of a generalized sentence. The difference is indicated by the fact that the operand in (5) is quoted and the operand in (6) is not. (5) is an abstraction of sentences, and (6) is an abstraction of propositions.

The relationship between the general proposition and individual ones can be designated as follows:

Jack.man :=: Jack is a man

Kathy.man :=: Kathy is a man

Unicorn.man :=: Ø

"Jack".man :=: Ø

where ":=:" means the identity of propositions.

Parallel to the case of sentential generalization, there are two ways of defining the denotation of the symbol "man" or "(λx:person)(x is a man)":

(a) It denotes a class of propositions sharing some common properties, i.e., the class of propositions in which a person is attributed with male humanity.

(b) It denotes a function operating on a symbol of the type person as its argument, and yields a proposition as its value.

In ADS, the second interpretation is adopted, and "man" is said to denote an attribute (or a propositional function). The expressions that denote an attribute are called predicates. Under this interpretation, the symbol "." denotes the function application operation, and "Jack.man" is equivalent to "man(Jack)" in mathematical notation.

The value of an attribute is undefined for an argument if it is not of the type specified as the range of the variable. Thus, in general, an attribute which is a propositional function is a partial function from symbols to propositions. It is very important to note that an attribute is not defined on objects but defined on symbols. The attribute, man, takes as its argument a symbol of type person, i.e., a symbol denoting a person, instead of a person itself.

4. The Concept of Type

In referencing "person" in "(λx:person)(x is a man)", we use the term "the type of the variable x". The type specifies what symbols the variable can assume as its value for substitution in the operand. We read "x:person" as "x is of the type person".

Here, we investigate the notion of type and the meaning of such an expression as "x is of the type person". First, we will show that the notion of type is closely related to the notion of significance or meaningfulness of a (declarative) sentence. Then, we will give a synonymous definition for an expression '_____ is of the type _____'.

In the previous section we have chosen to generalize the propositions expressed by (3) and (4) into a general proposition (propositional function), called man, which relates a person to the property of male humanity. The choice is based upon the hypothesis that any sentence of the form 'x is a man' is significant only when a symbol denoting a person is substituted for x. If a different assumption is made about the significance of sentences of the form 'x is a man', then a different propositional function must be taken as their generalization.

Consider the propositions expressed by the following sentences in addition to (3) and (4):

(7) "Kathy is a man"

(8) "Zeus is a man"

(9) ""Jack" is a man "

(10) "Boston is a man".

According to the hypothesis assumed for the definition of "man", the sentences (3), (4) and (7) are significant, and the rest are not. If we assume that (8) is also significant, then a different generalization such as ()x:person-like-entity)(x is a man) becomes necessary. If the sentences (3), (4) and (7) through (10) are to be considered significant, a possible generalization would be ()x:anything)(x is a man).

A generalization by the abstraction operator implies a commitment about the significance of sentences of a particular form. Such a commitment is represented by the type specification of the operator variable. Conversely, the type specification of the operator variable can be represented by a mechanism for testing the significance of an arbitrary sentence. For example, "Zeus" is of the type person if and only if "Zeus is a man" is significant, i.e.,

""Zeus" is of the type person" = /"Zeus is a man"/.

(Note that "Zeus is of the type person" is nonsignificant because Zeus is not a symbol.)

Therefore, a type specification is logically equivalent to a specification of what symbols produce significant sentences. It must be construed as a rule of symbol usage resembling a grammatical rule. The purpose of introducing the notion of type into a database model such as ADS is to eliminate the possibility of nonsignificant (or inconsistent) data from the database.

Now, we consider the meaning of the sentence:

(11) 'x' is of the type y

where x is an arbitrary symbol and y is an arbitrary predicate denoting an attribute (propositional function). We interpret (11) to mean that the symbol 'x' denotes an object having the attribute y, or equivalently, that 'x.y' is true. We need the quotes around x in (11) because the predicate 'is of the type y' requires a symbol as its subject. Since x is a variable, we can not use the double-quotes. Let us introduce a new notation:

(12) 'x..y' :==: ''x.y' is true',

:==: ''x' is of the type y',

then

(13) [x..y] = [x.y] if /'x.y'/,

= F otherwise.

'x..y' is read as 'x is y'.

For example, assume that "person" is defined as a predicate:

(14) person == (λx:animal)(x is rational).

Then,

"Zeus..person" :=: ""Zeus.person" is true",

:=: ""Zeus" is of the type person".

Similarly,

"Zeus..man" :=: ""Zeus.man" is true",

:=: ""Zeus" is of the type man".

Both "Zeus..person" and "Zeus..man" are false. While "Zeus.person" and "Zeus.man" are nonsignificant.

"x:person" in the definition of "man" is read as "the variable x is of the type person". Since by definition this is synonymous with "x..person", we may expect that the symbol ":" is replaceable either by ".." or by ".".

Consider

(15) man1 == (λx:anything)(x.person and x is a man)

(16) man2 == (λx:anything)(x..person and x is a man)

to demonstrate that this is not the case. Neither of these attributes is identical to the propositional function, man. To show this, consider the argument "Bambi", a name of an animal, for predicates man, man1 and man2:

"Bambi.man" :=: Ø

(because "Bambi" is not of the type person)

"Bambi.man1" :=: "Bambi.person and Bambi is a man"

:=: "Bambi is rational and Bambi is a man"

(because "Bambi" is of the type animal)

"Bambi.man2" :=: "Bambi..person and Bambi is a man"

:=: ""Bambi.person" is true and Bambi is a man"

:=: ""Bambi is person" is true and Bambi is a man".

Since "Bambi.man" denotes nothing, while "Bambi.man1" and "Bambi.man2" denotes the truth-value F, man is different from man1 and man2. In order to show the difference between man1 and man2, consider the argument "Zeus":

"Zeus.man1" :=: "Zeus.person and Zeus is a man"

:=: Ø

(because "Zeus.person" is non-significant)

"Zeus.man2" :=: "Zeus..person and Zeus ia a man"

:=: ""Zeus is person" is true and Zeus is a man".

While "Zeus.man1" denotes nothing, "Zeus.man2" denotes the truth-value F.

It is necessary to consider the symbol ":" as a part of the definition of the lambda operator "λ", and the symbol should not appear in any other context. For example, "(λx:anything)(x:person and x is a man)" should be considered as a syntactically incorrect expression. However, the symbol ":" can be replaced by a combination of significance testing and a conditional expression as follows:

'(λx:A)B' :=: '(λx)(/'x.A'/ → B; C)'

where A is a predicate, C is an arbitrary nonsignificant symbol, and 'a→b;c' means 'if a then b else c' in ADS.

5. Conclusions

We have shown that the concept of type in ADS is a special case of semantic abstraction and that a type specification is equivalent to a specification of a class of nonsignificant symbols. An important next step is to investigate the relevance of the concept of type in ADS to the notion of type in programming languages where type is used both for the elimination of syntactically correct but semantically incorrect expressions and for managing complex sets of data objects.

References

[1] Church, A., The Calculi of Lambda Conversion, Princeton University Press, 1941.

[2] Cox, Jr., J.R., "A Medical Information System Design Methodology," Annual Report to the National Center for Health Service Research, Department of Computer Science, Washington University, St. Louis, Missouri, June 1982.

[3] Cox, Jr., J.R., T.D. Kimura, and M.J. Stucki, "Design Studies Suggested by an Abstract Model for a Medical Information System," Proceedings of the Fourth Annual Symposium on Computer Applications in Medical Care, Washington, D.C., Nov. 1980, 1485-94.

[4] Frege, G., "On Sense and Reference," Translations from the Philosophical Writings of Gottlob Frege, eds., P. Geach and M. Black, Oxford:Blackwell, 1960.

[5] Kimura, T.D., W.D. Gillett, and J.R. Cox, Jr., "Abstract Database System (ADS): A Data Model Based on Abstraction of Symbols," Technical Report No. WUCS-82-12, Department of Computer Science, Washington University, St. Louis, Missouri, July 1982.

[6] Russell, B., "The Philosophy of Logical Atomism," Logic and Knowledge, R.C. Marsh, ed., New York:Putnam's Sons, 1956.

[7] Smith, J.M. and D.C.P. Smith, "Database Abstractions: Aggregation and Generalization," ACM Transactions on Database Systems 2:2 (June 1977), 105-33.