THE FORMAL SPECIFICATION OF SYSTEMS AND THE TEACHING OF LOGIC

Levis Zerpa Morloy
Universidad Central de Venezuela
zerpale@camelot.rect.ucv.ve

1. Introduction

It is well known that the use of computational tools in the teaching of logic motivates and can attract the attention of many students. If besides using computational tools as ***auxiliary*** resources in the logic course we also teach how to *apply logic and set theory to* ***develop*** *software*, the result of this strategy provides a powerful ***motivation*** for the student because in this way we can ***show how to use logic and set theory in order to improve the quality of the software***. On account of the great impact of the software in our society and the urgent need of developing safer and bug-free software, this application of logic can be very stimulating for the teaching of logic and set theory not only for the students of Computing and Sciences but also for those studying Humanities, depending on the sort of examples selected.

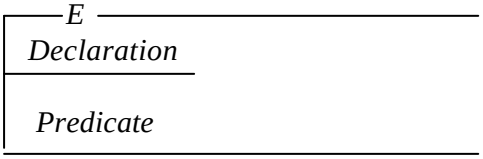
In this paper we informally examine one of the central subjects of what it is generally referred as formal methods in computer science: the ***formal specification of systems*** (= FSS). Within this field we only consider some operators of the ***schema calculus*** seen through a concrete example [1]: a simplified library system, *LibSys*. Our main objective is to show how *the FSS can be a high-yielding source of applications and examples of* ***formal reasoning with a very intuitive empirical interpretation*** for the students in the courses of Logic in Computing, Science, and Humanities.

2. What is the formal systems specification?

To specify a system is to describe its external behavior in a non-numerical way[2]. In other words: it is the description of ***what*** the system must do (or what we expect from it) without considering ***how*** will it do it (implementation details in a programming language). When we formalize the specification of a system by means of a formal language of specification based on the predicate calculus and set theory (as Z, see Spivey [1992]), we talk about a *formal specification* of that system. This technique allow us 1) to develop correct programs by construction and 2) to reason formally about the properties of the system. Both tasks are carried out through a systematic use of the first order predicate calculus and the theory of denumerable sets (and for these purposes both natural deduction and axiomatic systems are used). Unlike other developing methodologies, formal methods in computer science avoid ambiguities, contradictions, and make the documentation, support, debugging, and the consideration of all potential error cases in a given situation easier[3].

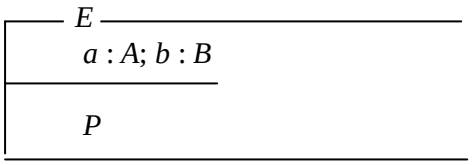
3. The schema calculus and the formal specification through an example

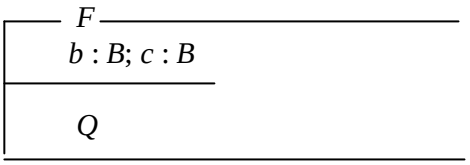
The fundamental unity of information in an FSS is the ***schema***. It consists of a first part where variables are declared (the *Declaration part*) and a second part with a set of constraints on those variables (the *Predicate part*). If "E" is the schema name we have E = [*Declaration* | *Predicate*] according to the linear notation or



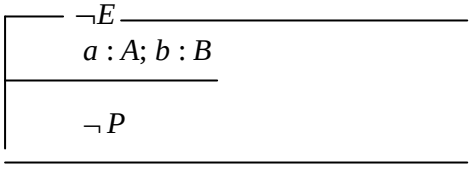
according to the bidimensional notation. Two different identifiers (with its type) and two different constraints are separated by a semicolon (";"). In the case of the predicate part, each semicolon is interpreted as a conjunction ("$\wedge$"). If in a diagram there is no predicate, we suppose the Boolean constant *true* as the predicate.

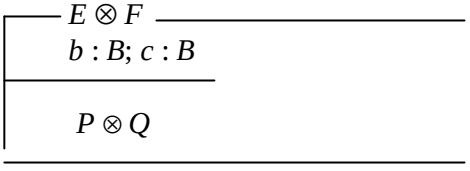
The operators of the schema calculus are $<\neg, \wedge, \vee, \forall, \exists, \backslash >$ and are defined as follows. Let be *E* and *F* two schemas



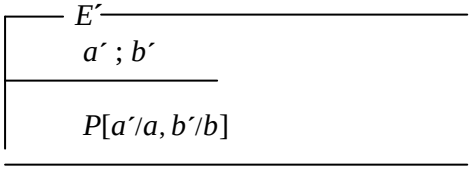


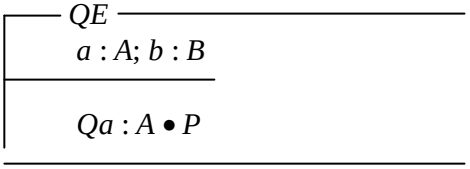
where a, b, c are variables of types A, B and C respectively, and P and Q are the predicates of schemas E and F respectively (E and F can share some variables or not). We have



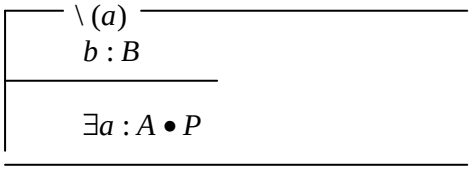


where $\otimes = \{\wedge, \vee\}$ and the ***negation of an schema*** E, $\neg E$, have a restriction: E must be normalized[4]. The schemas $E \wedge F$ and $E \vee F$ represent the ***conjunction and disjunction of schemas***. Each operation applies to the predicate part. A specially interesting case of the schema conjunction appears when a schema is included in the declarative part of another one, and this is called ***schema inclusion*** (this concept allows giving ***modularity*** to a formal specification[5]). The decoration and quantification of schemas are defined by





where $Q = \{\forall, \exists\}$. To describe the application upon a state of the system we must distinguish between the ***state prior*** to the application of the operation and the ***state after*** it. We distinguish both by adding an accent mark to all the components of the diagram that describes the later state, which is called ***decoration*** of a schema. $\forall E$ and $\exists E$ is the ***universal quantification and existencial quantification*** of the schema E which consists in quantifying one of the schema variables. A particular case of the existential quantification of schemas is the ***variables hiding*** operation (notice that only the variable which is not under the scope of the quantifier is declared):



In this way we can hide the variable quantified during the application of certain operations. Among other aspects of interest, *this operator allows a reinterpretation of the Ramsey statement in Epistemology* (as a methodological "elimination" of the theoretical terms in a scientific theory).

Let consider some notational conventions of the Z language. The ordered pair <x, y> is indicated by $x \alpha y$, $R > B$ es the range restricción of the relation R to the set B, # indicates cardinality. If A is a finite set, ℘ A is its power set. Si x is a variable, x? and x! indicate that x is an input or output operation, respectively. If E is a schema, ΔE indicates a state change, an initial state E and a final state E'. An exception: a partial function from X to Y is indicated by X $\rightarrow_p$ Y besides the original Z notation (by typographical reasons). With these conventions we will briefly consider the schemas that describe the states of a simplified system for a library, *LibSys*. The system has three basic types (maximal sets with a structure not specified at this moment): *Book*, *Copy* and *Reader* (we suppose that there are one or more copies of each book in the shelves). It includes a community of registered readers (the set *readers* of type ℘ *Reader*, that is, *readers* : ℘ *Reader*), a stock of books (partial function *stock* : *Copy* $\rightarrow_p$ *Book*), a set of copies on loan to readers (partial function *issued* : *Copy* $\rightarrow_p$ *Reader*) and the set

of copies on the shelves and available for loan (the set $shelved : \rightarrowtail Copy$). There is a maximum number of copies which any reader may have on loan at any time (constant $maxl : \mathbb{N}_1$). Two obvious conditions are the following:

- $shelved \cup \text{dom } issued = \text{dom } stock$
- $shelved \cap \text{dom } issued = \varnothing$

which forms the ***system invariant*** (ran $issued \subseteq readers$ too). In consequence, we have the following basic types and schemas: $[Book, Copy, User]$, $maxl : \mathbb{N}_1$.
Applying inclusion of schemas we can describe the "abstract states" of the system:

DbfLibrary

$stock : Copy \rightarrow_p Book$
$readers : \rightarrowtail Reader$

LoansLibrary

$issued : Copy \rightarrow_p User$
$shelved : \rightarrowtail Copy$

$shelved \cap \text{dom } issued = \varnothing$
$\forall u : readers \mid \#(issued \rhd \{u\}) \leq maxl$

Now we can include both schemas in the new schema *Library*:

Library

DbfLibrary
LoansLibrary

$shelved \cup \text{dom } issued = \text{dom } stock$
ran $issued \subseteq readers$

Naturally and as we have already shown, the loan of books is among the most important operations of the system. This operation is defined by the diagram $Issue_{Part}$ where the subindex indicates that it is a partial operation because it does not include error cases, that is to say, the cases when the operation can fail.

$Issue_{Part}$

$\Delta Library$
$e? : Copy$
$r? : Reader$

$e? \in shelved$
$r? \in readers$
$\#(issued \rhd \{u\}) \leq maxl$
$issued' = issued \cup \{e? \mapsto u?\}$
$stock' = stock$
$readers' = readers$

4. The formal reasoning about a specification as a source of motivating examples in the courses of Logic

One of the advantages of the schema calculus is that ***it facilitates the reasoning on the programs***. On the other hand, the *systematic consideration of the possible error cases* improves the quality of the system and the proof of its consistency guarantees that there are states that satisfy the system. Therefore, the system is consistent if it is satisfied by at least one state, then in order to proof its consistency it is enough to proof that there is an initialization state for the system, let us say a state *Library'* for which $stock' = \varnothing$ and $readers' = \varnothing$ This state is defined for the schema *IniLibrary*:

IniLibrary

Library'

$stock' = \varnothing$
$readers' = \varnothing$

Then, we proof the consistency of *LibSys* by means of the following ***initialization theorem***:

$$\vdash \exists Library' \cdot IniLibrary.$$

We can also reason on the conditions under which the defined operations can be applied***. The fact that the system has a quite obvious empiric interpretation made easier the formulation and demonstration of many theorems for the students.*** For example, if we define the operation of returning a book[6], then when we lend and return the same book the stock should remain the same. And if we lend a book the stock decreases, etc. Also, the design of the *databases* are also formally obtained with the schema calculus. Some more complex theorems can be conjectured on other aspects that require the definition of additional outlines, which will allow a ***refinement of the specification***. The pedagogic advantage of all this is that an ***intuitive counterpart*** for the formal reasoning is available, which is useful as a ***guide*** to the student[7]. ***In this way the logic and the set theory used to reason over the schemas allow to determine with accuracy the behavior of the system for any given state***. Also more and more ´´realistic´´ specifications can be obtained, which specificate with more details and accuracy the proposed objectives without this implying the introduction of undesirable features or the omission of the desirable ones. We insist: in all those cases the intuitive counterpart of the schema calculus is very useful for the logical reasoning on the system.

5. References

Burke and Foxley [1996]: Edmund Burke and Eric Foxley, *Logic and its Applications*, Prentice Hall, London, 1996.

Potter *et al.* [1996]: Ben Potter, Jane Sinclair and David Till, *An Introduction to Formal Specification and Z*, second edition, Prentice Hall, London, 1996.

Spivey [1992]: J. M. Spivey, *The Z Notation: A Reference Manual*, Prentice Hall, second edition, London, 1992.

Woodstock y Davies [1996]: Jim Woodcock and Jim Davies, *Using Z. Specification, refinement and proof*, Prentice Hall, London, 1996.

Notas

[1] There are already Logic textbooks in which FSS are exposed as a class subject: see Burke and Foxley [1996], ch. 6. The example appears in Potter *et al.* [1996], ch. 6.

[2] "*…with the emphasis on the logic of the system than the numeric predictions provided by, for example, differential equations.*" (Burke and Foxley [1996], p. 232).

[3] See Potter *et al.* [1996] and Woodcock and Davies [1996].

[4] See Woodcock and Davies [1996], p. 176.

[5] The same requirement that is made to routines in programming libraries.

[6] By means of a new schema $Return_{Part}$.

[7] Works as formal reasoning about familiar contexts in systems of Symbolic Artificial Intelligence and in a similar way as the "geometry of the problem" is useful as a ***pedagogical guide*** in other branches of mathematics (for example, integral calculus).