Circumscription in Data Logic for Data Type Specification

Leopoldo E. Bertossi

Abstract: In this paper we present a logical specification of data types. This specification is built upon a previously given specification for a data type that we want to extend by introducing new constructing objects and defined operations. The extended data type is specified by superposing a circumscription principle to some specification axioms. Circumscription formalizes in second-order logic some forms of (non-monotonic) common-sense reasoning in knowledge bases [11]. By means of circumscription we separate very clearly the first-order specification language (or axioms) from the specification mechanism. By showing the logical equivalence to Zhang's non-circumscriptive data type specification [17], we obtain the categoricity of our specification when we start from a categorical specification for the original data type.

Keywords: data type specification, circumscription, first-order logic, second-order logic, knowledge bases, non-monotonic reasoning.

1. Introduction

In this work we apply circumscription [11, 12] to the specification of data types, that is, we use it as a logical mechanism and principle which allows us to distinguish a many-sorted algebra, i.e. a structure consisting of a domain with individuals of different species and operations defined on the domain, among all possible many-sorted algebras that satisfy certain axioms expressed in a logical language. From this point of view, a datatype appears as a distinguished model of a specification written in a formal language of logic.

Some of the advantages of a logical specification have to do with its precision, its well-understood semantics, the clear separation between specification and implementation; and the possibility of using, at least in principle, theorem provers to effectively deduce properties of the intended datatype. One of the best apologies of logical specifications can be found in [13]. Most of the arguments given there in favour of logical specifications of relational databases also apply for logical specifications of datatypes.

Circumscription will be used in its second-order version [9] (in contrast to the original definition in terms of an infinite first-order schema [11]) : $C(\varphi; P)$, the circumscription of predicate P in the sentence φ, is defined as the second-order sentence $\varphi(P) \wedge \neg \exists X(\varphi(X) \wedge X \subset P)$. (The formula $X \subset P$ means that X is a proper subset of P. This formula might be replaced by the usual definition of the proper subset relation that does not involve second-order predicates). Then, the interpretation of predicate P in a model of this circumscription must have a minimal extension. We can see that $C(\varphi, P)$, by being stronger that φ, picks up some distinguished models of φ. This makes plausible the use of circumscription in data type specification.

In [17], Zhang introduces a data logic. This is a formal specification of data types in a many-sorted second-order logic that does not use circumscription. We will keep essentially the same axioms given by Zhang, but our use of circumscription will allow us to state them in a much simpler way. The application of circumscription in the specification need not be contained in the formal axioms, but rather can be left as a superposed principle or mechanism.

The reader will benefit from reading Zhang's paper [17], but our presentation will be self-contained except for some familiarity with circumscription (see also [7, 10]). In particular, we will use "prioritized circumscription" [9]: if $\varphi(P, Q)$ is a sentence containing P and Q among its predicates, $C(\varphi; P > Q)$ is the circumscription of P and Q in φ, where P has higher priority than Q in the minimization process.

Prioritized circumscription $C(\varphi; P > Q)$ is defined by the sentence $\varphi(P,Q) \wedge \neg\exists XY(\varphi(X,Y) \wedge (X,Y) \prec (P,Q))$. In this sentence, $(X,Y) \prec (P,Q)$ means that (X,Y) is strictly less than (P,Q) in the lexicographical order. Components in these pairs are compared according to the subset relation. The symbol "$\prec$" might be eliminated by using the straightforward definition of lexicographical order.

Usually, a many-sorted data type can be extended introducing a new domain of individuals (a new sort) and new operations whose ranges may be contained in the new domain. These new objects are called "constructors" of the new domain, or of the new specification [17] (see definition below). Other operations could be introduced by formal definitions in terms of constructors and objects within the original data type. This extension process raises the problem of obtaining a logical specification, in an expanded logical language, of the new data type starting from an specification of the original data type. It is precisely the specification of this extension what circumscription provides. For example, we could start with an algebra whose domain is the set of natural numbers, and extend it to a new data type with an additional domain of stacks of natural numbers and new operations such as "push" or "pop".

If the different domains or species are $S_1, \ldots, S_k$, with sorts 1, ..., k, then the individuals in these domains have the sorts $(1), \ldots, (k)$, respectively. The corresponding many-sorted logical language is endowed with variables of different sorts. If f is an operation from, say, $S_i \times S_j$ to S_l, then it has the sort (i,j,l). A typical structure or intepretation for this language has several domains. To make the application of circumscription easier and to avoid the formalism of many-sorted logic, we make a reduction of sorts [6] and use ordinary (one-sorted) second-order logic. Through sort reduction, we consider only one domain S and new predicates $S_1(\cdot), \ldots, S_k(\cdot)$ defined on it. It is possible to avoid the explicit reference to sorts in sentences by means of the relativization of the variables to the predicates $S_1, \ldots, S_k$. This natural translation allows us to use indistinctly both notations $c \in S_i$ and $S_i(c)$. For a treatment of circumscription in many-sorted languages, see [9].

2. Data logic

Suppose that finitely many sets $S_1, \ldots, S_k$ are given. They are called "initial domains", or "initial sorts". There might be also some operations. A typical operation goes from the cartesian product of some of the S_i to one of these domains. The next step consists in considering a new domain S_{k+1} and possibly new operations that involve this domain. The language has to be expanded by considering a new predicate for this domain and new symbols for constants and operations. In an example given before, we had the set of natural numbers as the initial domain, and the set of stacks of natural numbers, as the new domain. The empty stack could be a new distinguished individual and "push" could be a new operation. These new objects would be among the constructors of the new domain.

2.1 Definition: Let $S_1, \ldots, S_k$ denote the initial domains; and S_{k+1} , the new domain. The new individual constants c ($\in S_{k+1}$) and new operations g: $S_{i_1} \times \ldots \times S_{i_n} \to S_{k+1}$ ($i_j \leq k+1$) are called "constructors".

Constructors are among the new non-logical symbols in the expanded language. All the constructing constants as well as the ranges of the constructing operations belong to the new sort. Let $c_1, \ldots, c_l, g_1, \ldots, g_r$ be the different constructors of S_{k+1}. The sort of g_j will be given by g_j : $S_{j_1} \times \ldots \times S_{j_{m_j}} \times S_{j_{m_j+1}} \times \ldots \times S_{j_{n_j}} \to S_{k+1}$, with $j_1, \ldots, j_{m_j} \leq k$ and $m_j + 1 = \cdots = j_{n_j} = k+1$

As we said before, apart from the constructors we may introduce new operations by definition. This means that they can be considered as abbreviations and could be eliminated. If f is such an operation, Q_f will be a new predicate denoting the functional relation given by the graph of f. For simplicity, but also for being an interesting case, we will consider the situation where a new functional predicate Q_f is defined by primitive recursion.

Let Σ_1 be the set of sentences written in the language of second-order logic where only the symbols $S_1, \ldots, S_k$ and the non-logical symbols based on these predicates appear (we are thinking of a formal specification of the original data type). Now, let Σ_2 be a set of sentences written in an expansion of the former language where the new predicate S_{k+1} and other new non-logical symbols may appear. We want Σ_2 to be a formal specification of the extended data type:

2.2 Definition: We say that "Σ_2 is a data logic based on Σ_1" if

$$\Sigma_2 = \Sigma_1 \cup \{C(\mathbf{Ax}; S_{k+1} > Q_f)\} \tag{1}$$

In (1), $C(\mathbf{Ax}; S_{k+1} > Q_f)$ is the prioritized second-order circumscription [9] of predicates S_{k+1} and Q_f (S_{k+1} with higher priority than Q_f in the minimization process) in the conjunction $\mathbf{Ax}$ of the axioms **(a)**, **(b)**, **(c)** and **(d)** that follow:

a) Constructing Axiom "*constr*":

$$(\bigwedge_{t=1}^{l} S_{k+1}(c_t)) \wedge \bigwedge_{j=1}^{r} \forall x_1 \ldots \forall x_{n_j} (\bigwedge_{p=1}^{n_j} S_{j_p}(x_p) \rightarrow S_{k+1}(g_j(x_1, \ldots, x_{n_j})))$$

(notice that among the S_{j_p}'s there might be some S_{k+1}'s)

b) Distinguishing Axiom: the conjunction of all the sentences of the form

$$\forall x_1 \ldots x_{n_i} y_1 \ldots y_{n_j} (\bigwedge_{p=1}^{n_i} S_{i_p}(x_p) \wedge \bigwedge_{p=1}^{n_j} S_{j_p}(y_p) \rightarrow f_i(x_1, \ldots, x_{n_i}) \neq f_j(y_1, \ldots, y_{n_j})),$$

where $i \neq j$ and the f_i's are the constructors; either constants or operations.

c) Identifying Axiom: the conjunction of the following axioms (with the appropriately relativized universal quantification)

$$g_i(x_1, \ldots, x_{n_i}) = g_i(y_1, \ldots, y_{n_i}) \longleftrightarrow A_i(x_1, \ldots, x_{n_i}, y_1, \ldots, y_{n_i}),$$

where $A_i(\bar{x}, \bar{y})$ is a reflexive, symmetric, and transitive formula.

We need some motivation before giving axiom **(d)**. We want to introduce an axiom $\delta(Q_f)$ that, after an application of circumscription, defines functional predicate Q_f. In the general case [17], such an axiom would be of the form:

$$Q_f(\bar{x}) \leftrightarrow \varphi(\bar{x}). \tag{2}$$

We are not interested in the general case, but only in the case where Q_f is the graph of a new operation $f: S_{i_1} \times \ldots \times S_{i_{n+1}} \to S_{i_{n+2}}$ introduced by definition using primitive recursion. We are assuming that a minimal arithmetical structure (or the notion of successor) has been already defined.

The usual equations which define the operation f from operations g and h by primitive recursion are

$$\begin{array}{rcl} f(x_1, \ldots, x_n, 0) & = & g(x_1, \ldots, x_n) \\ f(x_1, \ldots, x_n, x_{n+1} + 1) & = & h(x_1, \ldots, x_n, x_{n+1}, f(x_1, \ldots, x_n, x_{n+1})) \end{array}$$

There is an explicit definition of f in second-order logic [3, 16]. Using second-order logic it is possible to express that Q_f is a minimal predicate that satisfies the recursion equations given above. Using this result we could give the explicit definition of predicate Q_f as a second-order axiom of the form (2). Nevertheless, we want all axioms (a) - (d) to be first-order, so that second-order logic appears only at the circumscriptive level. We will define predicate Q_f implicitly by means of induction-like axioms.

The implicit definition of Q_f expresses the fact that Q_f, the graph of f, is a minimal predicate satisfying some properties Φ and Ψ which are based on the primitive recursion equations given above [16]. With this intuition, our axiom $\delta(Q_f)$ will express the fact that Q_f is a relation on $S_{i_1} \times \ldots \times S_{i_{n+2}}$ with the desired properties Φ and Ψ, but not necessarily minimal (since we do not want to use second-order logic at this level). The ultimate definition of Q_f will be obtained in the final step, when we circumscribe in this list of axioms.

d) Defining Axiom:

the axiom $\delta(Q_f)$ for the definition of functional predicate Q_f is the sentence

$$\Phi(Q_f) \wedge \Psi(Q_f), \tag{3}$$

with Φ and Ψ defined by:

$$\begin{aligned}
\Phi(X) \quad &\equiv \quad \forall x_1 \in S_{i_1}, \ldots, x_n \in S_{i_n}, y \in S_{i_{n+1}}, z \in S_{k+1}[zero(y) \wedge \\
&\qquad Q_g(x_1, \ldots, x_n, z) \longrightarrow X(x_1, \ldots, x_n, y, z)] \\
\Psi(X) \quad &\equiv \quad \forall x_1 \in S_{i_1}, \ldots, x_n \in S_{i_n}, y, y_1 \in S_{i_{n+1}}, z, z_1 \in S_{k+1}[\\
&\qquad X(x_1, \ldots, x_n, y, z) \wedge succ(y) = y_1 \wedge Q_h(x_1, \ldots, x_n, y, z, z_1) \\
&\qquad \longrightarrow X(x_1, \ldots, x_n, y, z, z_1)]
\end{aligned}$$

($\forall x \in S(\ldots)$ is an abbreviation for $\forall x(S(x) \rightarrow \ldots)$)

Operations introduced by definition do not "construct" the new domain. On the contrary, they are constructed from the already constructed new domain S_{k+1} and other already existing objects. According to (1), we give S_{k+1} a higher priority in the circumscription process. We first minimize the new domain and then the relation Q_f to make it functional.

Notice that the conjunction Ax of axioms **(a)** - **(d)** is a first-order sentence. Second-order logic appears when we circumscribe some of the predicates in **Ax**. We could even think of replacing second-order circumscription by a first-order circumscription schema [11, 12]. Circumscription of S_{k+1} in the constructing axiom "*constr*" has the effect of producing a kind of induction axiom: S_{k+1} is a minimal domain that contains the new constructing constants and is closed under the constructing operations. The relationship between circumscription and induction was explored in [11].

Equality axioms **(b)** and **(c)** are essentially the axioms in Clark's identity theory [4]. The following example will show us, among other things, that axiom **(c)** is necessary to obtain the desired specification.

2.3 Example: (based on example 1 in [17]) Let us start with the empty algebra and the empty theory. Let us take as constructors 0 and *succ*. The functional predicate –or operation– introduced by primitive recursion will be $+$; and N will denote the new domain; corresponding to S_{k+1} above. In this case, Σ_2, the data logic –based on the empty theory– for the data type $\mathcal{N} = \langle IN, 0, succ, + \rangle$ of natural numbers is the circumscription $C(\mathbf{Ax}, N > +)$ of N and $+$, with higher priority for N, in the sentence $\mathbf{Ax}$ given by the conjunction of:

a) $N(0) \wedge \forall x(N(x) \rightarrow N(succ(x))))$

b) $\forall x(N(x) \rightarrow 0 \neq succ(x))$

c) $\forall xy(N(x) \wedge N(y) \longrightarrow succ(x) = succ(y) \leftrightarrow x = y)$

d) $\forall xyz \in N(y = 0 \wedge x = z \rightarrow +(x, y, z)) \wedge$

$\forall xyy_1zz_1 \in N(+(x, y, z) \wedge succ(y) = y_1 \wedge succ(z) = z_1 \longrightarrow +(x, y_1, z_1))$

In **(d)**, formula $y = 0$ corresponds to formula $zero(y)$; formula $x = z$, to formula $Q_g(x, z)$, where g is the identity function; and formula $succ(z) = z_1$, to formula $Q_h(x, y, z, z_1)$ with $h(x, y, z) := succ(z)$. $\Box$

We can see in the previous example that the injectivity of the successor function is obtained from the identifying axiom **(c)**. Our specification coincides with Peano's second-order axiomatization for the set of natural numbers with the successor function. By a theorem of Dedekind, we know that this axiomatization is categorical, i.e. all its models are isomorphic to the structure of natural numbers [5].

Since second-order logic appeared in the specification via circumscription $C(\mathbf{Ax}; S_{k+1} > Q_f)$ in (1), we would like to see its effects and the possibility of simplifying calculations. We have to take into account the places where the predicates S_{k+1} and Q_f appear in $\mathbf{Ax}$. We know that the syntactical form of the axioms is very important from two points of view: the satisfiability and the computability of circumscription [9]. In other words, our purpose is twofold. On one side, we want to show that circumscription produces the expected specification of the new data type. On the other side, we want to analyze circumscription (1) from a computational point of view by finding out on which parts of **Ax** this circumscription depends.

We will use a simple general fact:

2.4 Lemma: If φ does not contain predicate P and A is the sentence $\forall\bar{x}(P(\bar{x}) \rightarrow \varphi)$, then for every sentence ψ, $C(A \wedge \psi; P)$ is logically equivalent to $A \wedge C(\psi; P)$.

Proof: $C(A \wedge \psi; P)$ is defined by

$$A(P) \wedge \psi(P) \wedge \neg\exists X(X \subset P \wedge A(X) \wedge \psi(X)).$$

Since φ does not depend on P, $A(X)$ is $\forall\bar{x}(X(\bar{x}) \rightarrow \varphi)$. Then, we can write the circumscription above in the form

$$P \subseteq \varphi \wedge \psi(P) \wedge \neg\exists X(X \subset P \wedge X \subseteq \varphi \wedge \psi(X)).$$

It is easy to check that this sentence is logically equivalent to

$$P \subseteq \varphi \wedge \psi(P) \wedge \neg\exists X(X \subset P \wedge \psi(X)),$$

that is, to $A \wedge C(\psi; P)$. $\square$

As an immediate consequence of this lemma we obtain the following proposition that allows us to handle more easily some of the first-order quantifiers that are relativized to S_{k+1} in **Ax**.

2.5 Proposition: a) The distinguishing axiom can be driven out of the circumscription $C(\mathbf{Ax}; S_{k+1} > Q_f)$.

b) If S_{k+1} does not appear in the formulas A_i in (**c**) (as in example 2.3 and many other applications), then the identifying axioms can also be driven out of the circumscription. $\square$

2.6 Proposition: If Σ_2 is data logic based on Σ_1 and S_{k+1} does not appear in the identifying axiom **(c)**, then Σ_2 is logically equivalent to the conjunction of

i) Σ_1

ii) distinguishing axiom

iii) identifying axiom

iv) $C(\delta(Q_f); Q_f)$

v) $C(\text{“}constr\text{”}; S_{k+1})$.

Proof: By proposition 2.5, it suffices to show that

$$C(\text{“}constr\text{”} \wedge \delta(Q_f); S_{k+1} > Q_f) \tag{4}$$

is logically equivalent to the conjunction of (iv) and (v).

In fact, prioritized circumscription (4) is equivalent [9] to the following conjunction of non-prioritized circumscriptions:

$$C(\text{“}constr\text{”} \wedge \delta(Q_f); S_{k+1}; Q_f) \wedge C(\text{“}constr\text{”} \wedge \delta(Q_f); Q_f). \tag{5}$$

Since Q_f does not appear in “$constr$”, the second circumscription is equivalent to “$constr$” $\wedge C(\delta(Q_f); Q_f)$. We can forget “$constr$” in this conjunction because that axiom already appears in the first circumscription in (5). In this way we obtain (iv).

In the first circumscription in (5), Q_f is a variable predicate (in the process of minimization of S_{k+1}). That circumscription is defined by

$$\text{“}constr\text{”} \wedge \delta(S_{k+1}, Q_f) \wedge \neg\exists X \exists Y (X \subset S_{k+1} \wedge \text{“}constr(X)\text{”} \wedge \delta(X, Y)). \tag{6}$$

We will see that (6) is equivalent to

$$\delta(S_{k+1}, Q_f) \wedge \text{“}constr\text{”} \wedge \neg\exists X (X \subset S_{k+1} \wedge \text{“}constr(X)\text{”}). \tag{7}$$

Obviously (7) implies (6). That (6) implies (7) follows from the fact that $\delta(S_{k+1}, Q_f)$ is negative in S_{k+1} (S_{k+1} appears in the form $\forall x(x \in S_{k+1} \rightarrow$

$\ldots)$): if there were an X satisfying $X \subset S_{k+1} \wedge$ "$constr(X)$" in (7), then (X, Q_f) would falsify the last conjunct in (6).

We can drop $\delta(S_{k+1}, Q_f)$ in (6) because it is already present in (iv). In this way (7) becomes the definition of the circumscription in (v). $\square$

2.7 Remarks: a) Proposition 2.6 tells us that the new domain S_{k+1} is constructed inductively from the constructors; and function f can be obtained by circumscribing its graph Q_f in the primitive recursion relations $\delta(Q_f)$.

b) The hypothesis on the identifying axiom **(c)** is satisfied in the particular, but usual, case (see example 2.3) where it is a conjunction of "basic" sentences [4] of the form

$$g_i(x_1, \ldots, x_{n_i}) = g_i(y_1, \ldots, y_{n_i}) \longleftrightarrow (x_1 = y_1) \wedge \ldots \wedge (x_{n_i} = y_{n_i})$$

(note that in this case predicate S_{k+1} does not appear in the corresponding formula A_i)

c) Proposition 2.6 shows us that under the hypothesis on axiom **(c)**, our specification is equivalent to Zhang's axioms [17]. In particular, we obtain as a consequence of his "categoricity theorem" that data logic Σ_2 is categorical (i.e. it has a unique model up to isomorphism) if the original specification Σ_1 is already categorical.

3. Conclusions

1. We have seen that –under usual conditions– it is possible to give categorical data type specifications by extending previously given categorical specifications. The extensions are obtained as models of axioms written in data logic. Each extension process makes use of an expanded logical language.

2. The advantage of our approach lies in the simplicity of the specifying axioms. Actually, they are written in a first-order language. The final specification is obtained by circumscribing predicates in these first-order axioms. It is in this second step that second-order logic appears. In this way we can separate very clearly the first-order specification language from the circumscription that shows up as a superposed selection mechanism.

3. From our specification we can see that it is not necessary to use second-order logic in all its extension, but only a portion of it. Namely, only that weaker fragment that is needed for stating the circumscription axioms.

4. Our use of circumscription is very intuitive. There is only one application of circumscription where we minimize first the new domain and then, if necessary, the graphs of relations that are introduced by primitive recursion. In this sense, the circumscription policy for the logical specification contains only one simple declaration (see section 4 in [10]).

5. We presented a particular, but non trivial and not uninteresting case of data type specification in data logic. We restricted ourselves to the case where only one function is introduced by definition. If we had other predicates or functions introduced simultaneously or in some order of construction, we could use parallel and prioritized circumscription [9] in connection to these new objects in order to obtain the new specification.

6. The problem of data type specification, and many other specification problems in computer science, e.g. database transaction specifications [15], procedure specifications [8, 2], etc., have explicit and sometimes more hidden commonsense aspects. As in everyday life, we find in computer science many instances of commonsense reasoning [14] . In this regard, it is not surprising that John McCarthy's circumscription, originally developed for formalizing commonsense reasoning, has natural and useful application in datatype specification.

Acknowledgements: The author is indebted to Javier Pinto and Alvaro Campos for introducing him to circumscription and for illuminating conversations around a preliminary version of this work [1]; to Tania Bedrax for useful comments on the text; and to two anonymous referees for previous versions of this paper who helped the author with many interesting comments and suggestions. Part of this research was supported by DIUC Grant 91/04E.

References

[1] Bertossi, L. "Circumscription and Data Type Specification", in Proceedings of the X International Conference of the Chilean Computer Science Society, D. Fuller (ed.), 1989, 5-10.

[2] Borgida, A., Mylopoulos, J. & Reiter, R. "And Nothing Else Changes: The Frame Problem in Procedure Specifications" (Preliminary Version), to appear as Technical Report DCS TR 281, Dept. of Computer Science, Rutgers University, and Technical Report, Dept. of Computer Science, University of Toronto, 1992.

[3] Church, A. "An Introduction to Mathematical Logic", vol.1, Princeton University Press, 1956.

[4] Clark, K.L. "Negation as Failure", in 'Logic and Data Bases' Gallaire & Minker (eds.), Plenum Press, 1978, 293-322.

[5] Ebbinghaus, Flum & Thomas "Mathematical Logic", Springer, 1984.

[6] Enderton, H. "A Mathematical Introduction to Logic",Academic Press, 1972.

[7] Genesereth & Nilsson "Logical Foundations of Artificial Intelligence", Morgan Kaufmann, 1987.

[8] Meyer, B. "On Formalism in Specifications", IEEE Software, January, 1985, 6-26.

[9] Lifschitz, V. "Computing Circumscription", Proceedings 9th International Joint Conference on AI, 1985. Morgan Kaufmann Pubishers, 1985, 121-127.

[10] Lifschitz, V. "Circumscription", (draft dated Dec 11, 1990). To appear as a chapter in 'Handbook of Logic in AI and Logic Programming', Oxford University Presss, 1993?.

[11] McCarthy, J. "Circumscription - A Form of Non-Monotonic Reasoning", Artificial Intelligence, 13, 1980, 295-323.

[12] McCarthy, J. "Applications of Circumscription to Formalizing Commonsense Knowledge", Artificial Intelligence, 28, 1986, 89-116.

[13] Reiter, R. "Towards a Logical Reconstruction of Relational Database Theory", in 'On Conceptual Modelling:Perspectives from Artificial Intelligence, Databases and Programming Languages', Brodie, Mylopoulos & Schmidt (eds.), Springer, 1984, 191-233. Also in 'Readings in Artificial Intelligence and Databases', Mylopoulos & Brodie (eds.), Morgan Kaufmann Pub., 1989.

[14] Reiter, R. "Nonmonotonic Reasoning", Annual Reviews in Computer Science, vol.2, Annual Reviews Inc., 1987, 147-186.

[15] Reiter, R. "Formalizing Database Evolution in the Situation Calculus". Proceedings of the Fifth Generation Computer Systems, Tokyo, June 92. To appear.

[16] Robbin, J.W. "Mathematical Logic", Benjamin, 1969.

[17] Zhang Ming-Hua. "A Second Order Theory of Data Types", Acta Informatica, 25, 1988, 283-303.

Pontificia Universidad Catolica de Chile
Escuela de Ingenieria
Departamento de Ciencia de la Computacion
Casilla 306, Santiago 22, Chile.
e-mail: bertossi@ing.puc.cl