Reasons and Ways to Cope with a Spectrum of Logics

Alfio Martini Uwe Wolter
Edward Hermann Haeusler
Dept. Theoretical Computer Science - UFRGS - Porto Alegre - Brazil
University of Bergen - Dept. Informatics - Bergen, Norway
Dept. Informatics, PUC - Rio de Janeiro - Brasil

Abstract.[1] A key unresolved problem in modern software specification and development is the issue of formal interoperability, that is to say, how several pieces (specifications) can be connected and how their multiple (heterogenous) semantic descriptions impose constraint on each other. In this paper we introduce the concept of (logical) bridges and, after some motivational discussion, show how it works in a simple example taken from the field of algebraic specifications. The construction itself is parameterized in two essential ways: we work with a very general formalization of the idea of a logic (institutions) and build bridges on top of maps maps between them. At the end, we summarize our conclusions and point to interesting ways of both developing and applying this concept.

1 Introduction

Nowadays, it is widely recognizable that (modern) software specification and development takes place in a platform covering a wide range of rigorous semantic-based descriptions of diverse aspects or views of the system [16]. This includes, for example, both executable and non-executable formal specifications, models of concurrency and interaction, abstractions suitable for systems analysis and model checking, specification of security, dependability, and real-time requirements. As a consequence, a key unresolved problem is the interoperability problem, namely, how these several pieces could be connected and how their multiple and possibly heterogeneous semantic descriptions impose constraints on each other.

In this paper we introduce the concept of a *logical bridge* with which we can approach the issue of (formal) interoperability. These ideas capitalize on basic concepts found in the elegant and well-known theory of algebraic specifications [7, 3].

Our attention will be focused on those kind of formalisms having an underlying well defined logic. In this way, heterogenous specifications can be achieved once we are able to offer a construction linking theories from different logics.

2 Basic Concepts

Institutions were introduced in [9, 10] in order to describe in an uniform way several notions of a logical system used for specification purposes. An even stronger motivation was,

[1]Work partially supported by Fapergs and CNPq.

however, to give a mathematical semantics for Clear [4] independent of any particular underlying logic. Institutions concentrate on the model-theoretic aspects of a logic. Here a logic consists of a *syntax* and of a *semantics* that *fit together nicely*. The first observation is that syntax it is not only comprised by a signature and the corresponding language, set of sentences. The mapping which assigns to each signature the corresponding language is also part of the syntax. This association should also be at least monotonic. For instance, inclusion of vocabularies should be mapped to inclusion of languages. Secondly, semantics comprises a collection of possible interpretations and a relation of satisfaction between interpretations and sentences of the syntax for each given signature. Finally, syntax and semantics should fit together in a smooth way, i.e., satisfaction should be invariant under change of particular names for predicate and operate symbols, i.e., notation.

These ideas can be traced back at least to [2]. Imposing a categorical structure on the syntax, one sees the categorical concept of a functor lies at the heart of a formalization of syntax and semantics. All this is collected in the following definition.

An *institution* $\mathcal{I} = (\mathsf{Sign}, Sen, Mod, \models)$ consists of: a category Sign whose objects are called *signatures*; a functor $Sen : \mathsf{Sign} \to \mathsf{Set}$; giving for each signature a set whose elements are called *sentences* over that signature; a functor $Mod : \mathsf{Sign}^{op} \to \mathsf{Cat}$, giving for each signature Σ a category whose objects are called Σ-models, and whose arrows are called Σ-morphisms; and a function $\models$ associating to each signature Σ a relation $\models_\Sigma \subseteq \mid Mod(\Sigma) \mid \times Sen(\Sigma)$, called Σ-*satisfaction relation*, such that for each arrow $\phi : \Sigma_1 \to \Sigma_2$ in Sign the *satisfaction condition*

$$M_2 \models_{\Sigma_2} Sen(\phi)(\varphi_1) \iff Mod(\phi)(M_2) \models_{\Sigma_1} \varphi_1 \quad \text{(Satisfaction Condition)}$$

holds for any $M_2 \in \mid Mod(\Sigma_2) \mid$ and any $\varphi_1 \in Sen(\Sigma_1)$.

Almost every presentation of a logic is an institution. First order-logic, and its well-known fragments (both in unsorted and many-sorted cases), lambda-calculs, higher-order logics see [10], variant of temporal logics [1] and meta-logical frameworks, like rewriting logic, for instance [13], are some examples, to name a few.

Given a arbitrary institution $\mathcal{I}$, we can derive a number of concepts. A theory is a pair $T = \langle \Sigma, \Gamma \rangle$, where $\Sigma \in \mid \mathsf{Sign} \mid, \Gamma \subseteq Sen(\Sigma)$. Given theories $\langle \Sigma, \Gamma \rangle$ and $\langle \Sigma', \Gamma' \rangle$, $\phi : \langle \Sigma, \Gamma \rangle \to \langle \Sigma', \Gamma' \rangle$ is a *theory morphism* if $\Gamma' \models Sen(\phi)(\Gamma)$. This defines a category Th in the expected way. Now let, for each $\langle \Sigma, \Gamma \rangle$, $Mod_{\models}(\langle \Sigma, \Gamma \rangle)$ be the full subcategory induced by all models $M \in Mod(\Sigma)$, such that $M \models_\Sigma \Gamma$. Then, he satisfaction condition implies that for each $\phi : \Sigma \to \Sigma'$ the model functor $Mod : Mod(\Sigma') \to Mod(\Sigma)$ can be restricted to $Mod_{\models}(\langle \Sigma', \Gamma' \rangle) \to (\langle \Sigma, \Gamma \rangle)$, for each theory morphism $\phi : \langle \Sigma, \Gamma \rangle \to \Sigma', \Gamma' \rangle$ This means, globally, that we have the generalized model functor $Mod_{\models} : \mathsf{Th}^{op} \to \mathsf{Cat}$.

3 An Introduction to Bridges

The idea of this section is to introduce the reader into the way bridges work. It can be read from anyone with a reading knowledge of formal specifications. The nice point, as we will see, is that they work as a straightforward generalization of the usual idea of inclusion of specifications.

The first example might be considered a bit provocative, since it is very simple. However, at this point, we are interested in focusing on the construction itself, rather than in the relevance of the application itself.

To begin with, let us consider the classical example of the specification of lists of naturals, written in the sequel in the well-known Z-notation [15].

Since natural numbers are used everywhere, it is perfectly reasonable to assume that we have a single module specifying it, so that we can include it whenever we need it. The specification bellow is standard. Note that we se the *specification, theory*, and *module* to the denote the same object.

ThNat, *MSEqtl*

Sorts
  nat
Opns
  $0 :\to nat$
  $succ : nat \to nat$
  $plus : nat, nat \to nat$

$[x : nat]\ plus(x, 0) = x : nat$
$[x : nat, y : nat]\ plus(x, succ(y)) = succ(plus(x, y)) : nat$

Since we are interested in lists of naturals, we can extend the module above by a sort *list* and the usual operations of *header*, *tail*, and of *construction* of lists. The clause Include, as it is hinted by its name, includes the module *ThNat* in the module *ThList*.

ThList, *MSEqtl*

Include $[Nat, MSEqtl]$
Sorts
  $List$
Opns
  $emptylist :\to List$
  $head : List \to Nat$
  $tail : List \to List$
  $cons : Nat, List \to List$

$[x : nat, y : list] head(cons(x, l)) = x : nat$
$[x : nat, y : list] tail(cons(x, l)) = l : list$
$[\ \ ] tail(emptylist) = emptylist : list$

Note that, in the specification above, the operation *head* is underspecified, since we don't have a definition for $head(emptylist)$. This is mainly because, many-sorted algebras are unsuitable to model partial operations like this one. There are many efforts in the literature to cope with this burden, like error algebras, logic of partial algebras [14], and order-sorted logic [8], which will be considered in the next specification.

As an example of the use of this logic, consider the same specification of lists, but now using order-sorted logic.

ThList, *OSEqtl*

Bridge $[ThNat, MSEqtl]$

Sorts

$list, nelist$

Subsorts

$nat < nelist < list$

Opns

$emptylist :\rightarrow list$

$head : nelist \rightarrow nat$

$tail : nelist \rightarrow list$

$cons : nelist, list \rightarrow list$

$[x : nat, y : list] head(cons(x, l)) = x : nat$

$[x : nat, y : list] tail(cons(x, l)) = l : list$

Note that the the problem of undefinedness of *head* is now solved by introducing a new sort, *nelist* to denote non-empty lists.

However, note that now either we specify natural numbers again from scratch (well, we agree that it would not be that difficult) or we might consider reusing it, just the way it is written in many-sorted logic. In fact, we can proceed exactly like that (since, as we will see, we have a sound way to connect the two logics). However, since the imported module is actually written in another logic, we have the clause Bridge which denotes a codification or representation of the (old) specification into the new logic. Note that in this case, and as in most practical applications as well, the signature morphism component of the bridge is a simple inclusion of (coded) natural numbers. In fact, the "bridged" specification of natural numbers would only have the harmless declaration $nat < nat$, saying that *nat* is a subsort of itself.

It is also important to stress that the user, from the point of of the modeling of an application, can keep in mind the natural understanding of the specification in the old logic, since the bridge clause is completely defined once a map is established between the underlying institutions involved. But what is more important, he can keep in mind its many-sorted model understanding of the specification, which is much simpler that its order-sorted counterpart. Moreover, such step could be even achieved by a mechanical procedure, coded in some powerful meta-logical frameworks, like Rewriting Logic [13], for instance.

A more challenging situation is provided if we specify now lists using unsorted horn logic with equality, while still keeping in mind the desire to reuse the module of natural numbers in many-sorted logic. The specification of lists with a bridge for natural numbers is shown bellow:

In general, the well-definedness of the this situation is ensured by the fact that we have a map between the logics involved in the specification (see next section). Besides, note that the clause Bridge$[ThNat, MSEqtl]$, in this case, is actually equivalent to the following specification:

ThList, *UHEqtl*

Bridge $[Nat, MSEqtl]$
Sorts u
Pred $nat, list : u$
Opns $emptylist :\to u, head : u \to u, tail : u \to u, cons : u, u \to u$

$[\ ] list(emptylist)$
$[x, y : u] nat(x), list(y) \Rightarrow list(cons(x, y)), nat(x), list(y) \Rightarrow nat(head(cons(x, y))$
$[x, y : u] nat(x), list(y) \Rightarrow list(tail(cons(x, y))$
$[\ ] tail(emptylist) = emptylist$
$[x, y : u] nat(x), list(y) \Rightarrow head(cons(x, y)) = x, nat(x), list(y) \Rightarrow tail(cons(x, y)) = y$

ThNat, *UHEqtl*

Sorts u
Pred $nat : u$
Opns $0 :\to u, succ : u \to u, plus : u, u \to u$

$[\ ] nat(0), [x : u] nat(x) \Rightarrow nat(succ(x)))$
$[x, y : u] nat(x), nat(y) \Rightarrow nat(plus(x, y)), [x : u] nat(x) \Rightarrow plus(x, 0) = x$
$[x, y : u] nat(x), nat(y) \Rightarrow plus(x, succ(y)) = succ(plus(x, y))$

4 Maps between Logics

A very flexible way to relate logics can be given by the following concept, due to [12]. It says that the target logic $\mathcal{I}'$ is rich enough to model the semantic requirements of the source logic $\mathcal{I}$.

There are many situations where we can translate signatures and sentences of an institution $\mathcal{I}$ into signatures and sentences of another institution $\mathcal{I}'$, but where only subclasses of the corresponding model classes in $\mathcal{I}'$ can be translated back into models of $\mathcal{I}$. Fortunately, we are able in most cases to axiomatize these subclasses within $\mathcal{I}'$. This is the central idea of the following

Let $\mathcal{I}$ and $\mathcal{I}'$ be institutions. A *simple map of institutions* $(\Phi, \Psi, \alpha, \beta) : \mathcal{I} \to \mathcal{I}'$ is given by a functor $\Phi : \mathsf{Sign} \to \mathsf{Sign}'$; a functor $\Psi : \mathsf{Sign} \to \mathsf{Th}'$ such that $\Psi(\Sigma) = \langle \Phi(\Sigma), \varnothing'_\Sigma \rangle$; a natural transformation $\alpha : Sen \Rightarrow \Phi;\ Sen' : \mathsf{Sign} \to \mathsf{Set}$, and a natural transformation $\beta : \Psi^{op};\ Mod'_{\models} \Rightarrow Mod : \mathsf{Sign}^{op} \to \mathsf{Cat}$, such that the *simple map of institutions condition*

$$\beta(\Sigma)(M') \models_\Sigma \varphi \Longleftrightarrow M' \models'_{\Phi(\Sigma)} \alpha(\Sigma)(\varphi) \quad \text{(Simple Condition)}$$

holds for each $\Sigma \in \mid \mathsf{Sign} \mid$, $M' \in \mid Mod'_{\models}(\Psi(\Sigma)) \mid$, and $\varphi \in Sen(\Sigma)$.

$$\begin{tikzcd} Mod(\Sigma) \arrow[r, leftrightarrow, "\models_\Sigma"] & Sen(\Sigma) \arrow[d, "\alpha(\Sigma)"] & \mathsf{Sign} \arrow[d, "\Psi"] \\ Mod'_{\models}(\Psi(\Sigma)) \arrow[u, "\beta(\Sigma)"] \arrow[r, leftrightarrow, "\models'_{\Phi(\Sigma)}"'] & Sen'(\Phi(\Sigma)) & \mathsf{Th}' \end{tikzcd}$$

□

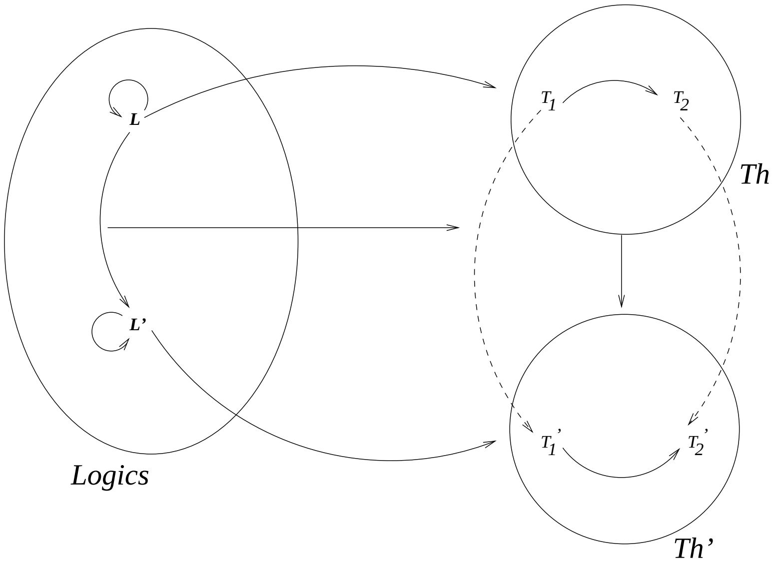

Figure 1: Bridges between theories

It is not difficult to see that simple maps compose. Then institutions as objects and simple maps as morphisms define the category Logics.

Proposition[Indexed Category] Let $(\Phi, \Psi, \alpha, \beta) : \mathcal{I} \to \mathcal{I}'$ be a simple map. Then the assignments $\mathcal{I} \mapsto \mathsf{Th}(\mathcal{I})$ $(\Phi, \Psi, \alpha, \beta) : \mathcal{I} \to \mathcal{I} \mapsto \Psi_{\models} : \mathsf{Th}(\mathcal{I}) \to \mathsf{Th}(\mathcal{I}')$ defines an indexed category $\mathsf{Th} : \mathsf{Logics} \to \mathsf{Cat}.\Box$

The following example is adapted from [5].

Consider the institutions of many-sorted equational logic *MSEqtl* and of unsorted Horn logic with equality $USHorn^{=}$, respectively. Firstly, the functor $\Phi : \mathsf{Sign}_{MSEqtl} \to \mathsf{Sign}_{USHorn^{=}}$ is given by translating any many sorted signature $\Sigma = (S, OP)$ into the Horn signature (OP, P_S) with unary typing predicates $P_S \stackrel{def}{=} \{\pi_s \mid s \in S\}$. Secondly, any equation in context $\varphi = (X \vdash t = u) \in Sen_{MSEqtl}(\Sigma)$, where X is an S-sorted family of variables, can be translated into the one-sorted equivalent one $(\forall x_1, \ldots, x_n : \pi_{s_1}(x_1) \wedge \ldots \wedge \pi_{s_n}(x_n) \to t = u) \in Sen_{USHorn^{=}}(OP, P_S)$, i.e., we actually have a natural transformation $\alpha : Sen_{MSEqtl} \Rightarrow \Phi;\ Sen_{USHorn^{=}}$. Thirdly, the introduction of typing predicates allows to extract a Σ-algebra A out of any (OP, P_S)-structure M where $A_s \stackrel{def}{=} \{m \mid \pi_s^M(m)\}$ and $op^A : A_{s_1} \times \cdots \times A_{s_n} \to A_s$ is the corresponding restriction of $op^M : M \times \cdots \times M \to M$. In general, we obtain by this procedure partial operations op^A, so that we have to restrict the translation to those (OP, P_S)-structures M that represent total Σ-algebras, i.e., structures M satisfying the set of additional axioms

$$\varnothing'_\Sigma = \{\forall x_1, \ldots, x_n : \pi_{s_1}(x_1) \wedge \ldots \wedge \pi_{s_n}(x_n) \longrightarrow \pi_s(op(x_1, \ldots, x_n)) \mid op \in OP\}.$$

This gives rise to a functor $\Psi : \mathsf{Sign}_{MSEqtl} \to \mathsf{Th}_{USHorn^{=}}$ and a natural transformation $\beta : \Psi^{op};\ Mod_{USHorn^{=}\models} \Rightarrow Mod_{MSEqtl}$. Note, finally, that the typing premise $\pi_{s_1}(x_1) \wedge \ldots \wedge \pi_{s_n}(x_n)$ in $\alpha(\Sigma)(\varphi)$ ensures that all first order representatives of an algebra A will satisfy $\alpha(\Sigma)(\varphi)$ if A satisfies φ. The implication into the other direction would be valid even if we would omit the typing premise. $\Box$

5 Bridges

Building on the above notion, we can now link in a natural way theories from these different logics. This is provided by the concept of a *bridge* right below.

Definition[Bridges] Let $(\Phi, \Psi, \alpha, \beta) : \mathcal{I} \to \mathcal{I}'$ be a simple map, and $\langle \Sigma, \Gamma \rangle$, $\langle \Sigma', \Gamma' \rangle$ theories from Th and Th'. Then a *bridge* between $\langle \Sigma, \Gamma \rangle$ and $\langle \Sigma', \Gamma' \rangle$ is a an arrow $\psi : \langle \Sigma, \Gamma \rangle \to \langle \Sigma', \Gamma' \rangle$ such that $\psi : \Phi(\Sigma) \to \Sigma'$ is an arrow in Sign' and such that $\Gamma' \models_{\Sigma'} \mathit{Sen}(\Phi(\phi))(\alpha(\Sigma)(\Gamma) \cup \varnothing'_{\Sigma})$.□

Bridges compose and build a category which is on top of the category Logics. It is essentially an indexed construction. This situation is best depicted in Figure 1. Each logic defines a category of theories in which diagrams model structured specifications. Now, each map induces a corresponding translation between the corresponding categories of theories. The solid arrows that link two given theories (inside the circles) represent the usual notion of theory morphisms, while the dotted ones denote bridges. Note that, in the figure, the loops denote identity maps, i.e., and by translating them, we recover the special case of specification in one fixed, arbitrary logic. This discussion is formalized below:

Proposition[Category of Bridges] Theories as objects (from arbitrary institutions) and bridges as arrows define the category $\mathsf{Bridges}$.□

Definition[Grothendieck Construction] Given a functor $F : \mathsf{C} \to \mathsf{Cat}$, the Grothendieck construction constructs the opfibration induced by F, a category $\mathsf{G}(\mathsf{C}, \mathsf{F})$, defined as follows:

- An object of $\mathsf{G}(\mathsf{C}, \mathsf{F})$ is a pair $\langle A, x \rangle$, where A is an object of C and x an object of $F(A)$.
- An arrow $\langle f, u \rangle : \langle A, x \rangle \to \langle A', x' \rangle$ has $f : A \to A'$ an arrow of C and $u : F\, f(x) \to x'$ an arrow of $F(A')$ (note that, by definition, $F;\ f(x)$ is an object of $F(A')$).
- If $\langle f, u \rangle : \langle A, x \rangle \to \langle A', x' \rangle$ and $\langle g, v \rangle : \langle A', x' \rangle \to \langle A'', x'' \rangle$, then $\langle f, u \rangle;\ \langle g, v \rangle : \langle A, x \rangle \to \langle A'', x'' \rangle$ is defined as:

  $$\langle f, u \rangle;\ \langle g, v \rangle = \langle F\ g(u);\ v, f;\ g \rangle$$

Theorem[Caracterization of $\mathsf{Bridges}$] The category $\mathsf{Bridges}$ is essentially the Grothendieck (flattened) category $\mathsf{G}(\mathsf{Logics}, \mathit{Th})$, where $\mathit{Th} : \mathsf{Logics} \to \mathsf{Cat}$.□

6 Concluding Remarks & Further Work

A related work can be found in the *extra-theory* morphism concept of Diaconescu [6]. Some initial examinations [11] make us expect that our proposal is simpler and more general, and a a detailed study should be given elsewhere. In general, bridges support formal interoperability in the sense that theories (specifications) from other logics can be reused such that logical consequence is preserved. Besides, the necessary coding of imported theories is defined once and for all, as long as a map between the underlying logics is established. We also believe that this approach has great benefits from the software engineering point of view, not only because of its expressive power, but also due to the fact that very costly implementation efforts can be avoided (by reuse of tools like theorem provers).

As further work, we believe that the concept of *bridges* can be very promising for giving, for instance, a smooth treatment of the Hoare Calculus in the semantics of programming languages. It has the feature of including the theory of the Data-Types involved in the programming language as logical (Full First-Order Logical) theories in the whole deductive system (and hence the logical entailment).

In another context, we beginnin to study this model in order to enablee the integration in a very general and smooth way, process languages (like CCS, CSP) that are very suitable for state-driven features of a given system, with more adequate logics for data features, as for instance, first order logic, or the logics usually used in the realm of algebraic specifications.

References

[1] M . Arrais and J. L. Fiadeiro. Unifying theories in different institutions. In *Recent Trends in Type Specification*, pages 81–101. Springer, LNCS 1130, 1996.

[2] K. J.. Bairwise. Axioms for Abstract Model Theory. *Annals of Mathematical Logic*, 7:221–265, 1974.

[3] R. M. Burstall and J. A. Goguen. Putting theories together to make specifications. In *Proc. Int. Conf. Artificial Intelligence*, 1977.

[4] R. M. Burstall and J. A. Goguen. The semantics of Clear, a specification language. In D. Bjøner, editor, *Abstract Software Specification, Proc. 1979 Copenhagen Winter School, LNCS 86*, pages 292–332. Springer, 1980.

[5] M. Cerioli and J. Meseguer. May i borrow your logic? (transporting logical structures along maps). *Theoretical Computer Science*, 173(2):311–347, 1997.

[6] R. Diaconescu. Extra-theory morphisms in institutions: logical semantics for multi-paradigm languages. *Journal Applied Categorical Structures*, 1998. To appear.

[7] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics*, volume 6 of *EATCS Monographs on Theoretical Computer Science*. Springer, Berlin, 1985.

[8] J. Goguen and J. Meseguer. Order-sorted algebra I: equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theoretical Computer Science*, 105:217–273, 1992.

[9] J. A. Goguen and R. M. Burstall. Introducing institutions. In *Proc. Logics of Programming Workshop*, pages 221 – 256. Springer LNCS 164, 1984.

[10] J. A. Goguen and R. M. Burstall. Institutions: Abstract Model Theory for Specification and Programming. *Journal of the ACM*, 39(1):95–146, January 1992.

[11] A. Martini. *Relating Arrows between Institutions in a Categorical Framework*. PhD thesis, Tecnhical University of Berlin, 1999. Dept. of Computer Science.

[12] J. Meseguer. General logics. In H.-D. Ebbinghaus et. al., editor, *Logic colloquium '87*, pages 275–329. Elsevier Science Publishers B. V.,North Holland, 1989.

[13] J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *TCS*, 96:73–155, 1992.

[14] H. Reichel. *Initial Computability, Algebraic Specifications, and Partial Algebras*. Oxford University Press, Oxford, 1987.

[15] J . M. Spivey. *Understanding Z: A Specification language and its formal semantics*, volume 3 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, 1988.

[16] A. Tarlecki. Moving between logical systems. In *Recent Trends in Data Type Specification*, pages 478–502. Springer, LNCS 1130, 1996.