On implementations of loose abstract data type specifications
and their vertical composition

Christoph Beierle
TK LILOG, IBM Deutschland GmbH
Postfach 80 08 80, D-7000 Stuttgart 1
EARN: BEIERLE at DS@LILOG

Angelika Voß
GMD, Forschungsgruppe Expertensysteme
Postfach 12 40, D-5205 St. Augustin 1
USENET: AVOS%GMDXPS at GMDZI

Abstract: In an approach for the implementation of loose abstract data type
specifications that completely distinguishes between the syntactical level of
specifications and the semantical level of models, vertical implementation composition
is defined compatibly on both levels. Implementations have signatures, models, and
sentences where the latter also include hidden components, which allows for useful
normal form results. We illustrate the stepwise development of implementations as well
as their composition by some examples and describe the incorporation of the concept
into an integrated software development and verification system.

1. Introduction

In the early days of abstract data types merely fixed ADT specifications with only isomorphic
models were studied. Later on, so-called loose approaches were suggested where one considers not
only the initial or terminal model of a specification but all models. As one of its main
advantages a loose approach is better suited to capture the process of software development: One
can start with a small and still vague specification with many different models, and then refine
such a specification gradually by adding new axioms, sorts, and operations, thereby restricting
the class of admissible models. During this process, lower level constructive definition
techniques may be used to refine the higher level axiomatic definitions so that one finally
arrives at a concrete problem solution, which could be a program or a functional prototype.

An implementation relation between loose specifications should reflect this refinement scenario:
among the many different models of the source and target specifications one should be able to
develop those of interest by gradually refining the implementation so that the set of models is
restricted accordingly. Our implementation concept introduced in [BE 85a] generalizes the
concept for implementations of loose specifications proposed by Sauvella and Wirsing in [SW 82],
which in turn generalizes the fixed case (e.g. [GTW 78], [Hc 82], [KMP 82], [Ga 83]). By using
the notion of institution ([GB 83]) our approach abstracts from the types of sentences used in
the underlying ADT specification method.

One of the central problems when dealing with implementations is their composability. In our
concept of implementation specifications the composition of implementations can be defined both
on the syntactical level of specifications and on the semantical level of models. Both levels
are closed under their composition operations which are associative. In particular, by using a
strong normal result we show that syntactical and semantical compositions are compatible with
each other.

In Section 2 we summarize the basic idea of our implementation concept as given in [BE 85a],
elaborate the requirements a composition operation should fulfill, and briefly state the
assumptions about the underlying loose ADT specifications. In Section 3 we introduce the
institution of implementation specifications without hidden components, and in Section 4 we
extend this institution by introducing hidden parts. Section 5 contains our normal form theorem,
and in Section 6 we develop syntactical and semantical composition operations and show their
compatibility. Section 7 describes the incorporation of our concept into an integrated software
development and verification system, and Section 8 contains a summary and a comparison.
Acknowledgements: We would like to thank Martin Wirsing and Jörg Siekmann for some valuable discussions and suggestions. This work was performed at the Universität Kaiserslautern and was supported in part by the Bundesministerium für Forschung und Technologie (IT 830353) and the Deutsche Forschungsgemeinschaft (SFB 314).

2. Implementation specifications: Basic idea and requirements for their composition

As compared to fixed specifications, in the loose case we still have specifications, signatures, signature morphisms, etc., the essential difference lying in the number of models being considered. Therefore, an implementation for loose specifications should at least consist of an abstract specification, a concrete specification, and a signature morphism translating the abstract signature to the (possibly extended) concrete signature. Since a concrete specification can always be extended before giving the implementation, we will choose the technically simpler approach and omit any extension of the concrete specification as part of the implementation.

In [SW 82] Sannella and Wirsing require for every concrete model some abstract model and an abstraction function connecting them. If such a complete set of triples exists, the concrete specification is said to implement the abstract one, otherwise it does not. This is an implicit, non-constructive approach which gives no room for a notion of refinement between implementations since there is no way to characterize and restrict the set of triples — e.g., by constraints on the concrete or abstract models — any further.

Since the idea of loose specifications is to consider at first an arbitrary large set of models and to restrict this set stepwise by refining the specification, we think the adequate idea of implementations between loose specifications is to accept all meaningful combinations of an abstract model, a concrete model, and an abstraction function and to restrict them stepwise by refining the implementation.

To realize these ideas we introduce the notion of implementation models: A simple implementation \(<SP_a, o, SP_c>\) consisting of an abstract specification \(SP_a\), a concrete one \(SP_c\), and a signature translation \(o\) between them denotes the set of all triples consisting of an abstract model \(A_a\), a concrete one \(A_c\), and an abstraction function \(o\) from the concrete to the abstract model. Such a triple \(<A_c, o, A_a>\) is called an implementation model. As in the fixed case, the abstraction function may be partially defined and it must be surjective and homomorphic. (Note that in both cases the first component contains the source and the third component the target of the function in the middle component.)

A refinement between implementations should restrict the set of implementation models which can be done componentwise by restricting the sets of abstract models, of concrete models, and of abstraction functions. In the framework of loose specifications sets of models — like the abstract and the concrete ones — are restricted by adding sentences to the respective specification. To apply this technique to implementations we view abstraction functions, which operate on both concrete and abstract carriers, as algebra operations from concrete to abstract sorts. These operations can be restricted as usual by adding sentences over both the concrete and the abstract signatures extended by the abstraction operation names. Thus we admit arbitrary sentences over the abstract and the concrete signatures extended by the abstraction operation names — later on we will extend this vocabulary by arbitrary hidden sorts and operations. These sentences will be called implementation sentences.

Adding a set \(IE\) of implementation sentences to a simple implementation \(IL = <SP_a, o, SP_c>\) we obtain an implementation specification \(ISP = <IL, IE>\) that denotes all implementation models of \(IL\) which satisfy \(IE\). Analogously to specifications which consist of a signature in the simplest case, a simple implementation like \(IL\) will also be called an implementation signature.

An implementation should be refinable by adding more implementation sentences to it and thus reducing the class of implementation models. This idea is extended analogously to loose ATP specifications by admitting a change of signature: There, a specification morphism is a