A notion of implementation for the specification language

\textit{OBSCURE}

Thomas Lehmann
Universität des Saarlandes
FB14 Informatik
D-6600 Saarbrücken
e-mail: lehmann@cs.uni-sb.de

1 Introduction

In the beginning of the seventies abstract data types have been introduced with the goal to enhance security in the software development process. This work has been strongly influenced by the data type concepts in programming languages such as CLU ([LZ74]) and by the pioneering paper of Hoare ([Hoa72]). A lot of theoretical investigation has been made in the field of algebraic specifications of abstract data types starting with the initial algebra semantics of the ADJ group ([GTW78]). Alternatively other semantics for algebraic specifications have been studied, e.g. final semantics ([Wan79, Kam83]) or observational abstraction mechanisms ([GGM76]). Apart from the algebraic approach several authors suggested more constructive specification methods ([Car80, Kin84, Loe87]). Similarly to the development of programming languages, so called specification languages have been developed based on these methods, beginning with CLEAR ([BG80, San84]) and followed by ACT ONE ([EM85]), ACT TWO ([Fey88, EM90]), OBJ2 ([FGJM85]), OBJ3 ([GW88]), PLUSS ([Gau84, BGM87]) and ASL ([Wir86]).

In order to describe the process of developing concrete programs from abstract software specifications in the sense of stepwise refinement, several notions of implementation appeared in the literature. They study the question under which circumstances a specification is implemented by another one. The first concept proposed by [GTW78] for algebraic specifications has been extended by [EKMP82] and [Ehr82]. The approaches of [EK82] and [Lip83] considered implementations of parameterized specifications. More recent papers deal with notions of implementation in the context of loose specifications and mainly rely on the refinement concept of [SW83]. Examples are among others the work of Sannella and Tarlecki ([ST88]), Beierle and Vöß ([BV85]), Schoett ([Sch86]) and Hennicker ([Hen89]).

The notion of implementation we will present here is based on the specification language \textit{OBSCURE} whose semantics has been precisely described in [LL87]. \textit{OBSCURE} provides tools to build up modules. The semantical behaviour of such a module is characterized by a function mapping algebras over its imported signature to algebras over its exported signature. The semantics is called \textit{fixed} (in contrast with \textit{loose} semantics), because for each imported algebra at most one exported algebra is admitted, whereas in a loose approach the semantical function maps an imported algebra to a \textit{class} of exported algebras. Therefore the notion of implementation presented in this paper — although related to various approaches known from the literature — does not incorporate the refinement concept of [SW83] so far. In [Leh90] we discuss a possible extension of \textit{OBSCURE} to loose semantics and transfer our notion of implementation to the new situation.

An implementation in our sense relates two \textit{OBSCURE} modules. On the level of signatures this relation is characterized by morphisms connecting the imported and exported signatures of the module to be implemented with the corresponding signatures of the implementing module. On the level of algebras an...
implementation describes how a homomorphic relation between algebras over the imported signatures of the implementing and the implemented module can be extended to the corresponding exported algebras. The idea of such homomorphic relations, that will be called representations henceforth, occurs in many papers on abstract implementations and can already be found in the abstraction functions of [Hoa72].

We are especially interested in two different methods to combine implementations. The vertical composition of two implementations shows how from an implementation of a module \( m \) by a module \( m' \) and from an implementation of this module \( m' \) by a module \( m'' \) an implementation of \( m \) by \( m'' \) can be constructed. Horizontal composition relates implementations with the underlying specification language. Here we assume to be given modules \( m_1, \ldots, m_r \) that can be combined to a module \( m \) using the specification building operations of \( \text{OBSCURE} \). If moreover \( m_1', \ldots, m_r' \) are modules implementing \( m_1, \ldots, m_r \), horizontal composition of implementations means that the combination \( m' \) of the modules \( m_1', \ldots, m_r' \) yields an implementation of the module \( m \).

The paper is organized as follows. After a short introduction recalling several features of the specification language \( \text{OBSCURE} \) (Section 2), we define the notion of an implementation in Section 3 and illustrate the concept on a running example. Section 4 lists the main results, especially the vertical and horizontal composition properties, and provides an example using some of these results.

# 2 The specification language \( \text{OBSCURE} \)

This section presents a short introduction into the specification language \( \text{OBSCURE} \). A more detailed version may be found in [LL87]. For the notions of signature, signature morphism and algebra we refer to the classical work of [EM85]. When dealing with formulas we always have formulas of first order predicate logic in mind with equality as the only predicate symbol. Given a signature \( \Sigma \) we denote its underlying set of sorts by \( \text{SORT}(\Sigma) \) and the set of first order formulas over \( \Sigma \) containing variables from an (implicitly given) \( \text{SORT}(\Sigma) \)-sorted family \( X \) of infinite sets by \( \text{WFF}(\Sigma) \).

For a formula \( w \) over \( \Sigma \) with variables from \( X \), a \( \Sigma \)-algebra \( A \) and a (variable) assignment \( \alpha : X \rightarrow A \) the satisfaction relation \( A, \alpha \models w \) is defined as usual; we write \( A \models w \) if \( A, \alpha \models w \) holds for all assignments \( \alpha : X \rightarrow A \).

It is our goal to describe the set \( \text{SPEC} \) of all \( \text{OBSCURE} \) specifications which is characterized by a signature function \( \mathcal{S} \), assigning to each module its imported and exported signatures, and by a so called meaning function \( \mathcal{M} \). \( \mathcal{M} \) describes the semantical behaviour of the modules from \( \text{SPEC} \), i.e. for each module \( m \) its semantics \( \mathcal{M}(m) \) is a (possibly partial) function mapping algebras over the imported signature of \( m \) to algebras over its exported signature.

To define the set \( \text{SPEC} \) and the functions \( \mathcal{S} \) and \( \mathcal{M} \) we start from a given set \( \text{AtSPEC} \) of so called atomic specifications (atomic modules) and two functions \( \mathcal{S}^0 \) and \( \mathcal{M}^0 \) defined on \( \text{AtSPEC} \). The function \( \mathcal{S}^0 \) maps each \( m \in \text{AtSPEC} \) to a pair of signatures \( \mathcal{S}^0(m) = (\mathcal{S}^0_i(m), \mathcal{S}^0_e(m)) \). \( \mathcal{S}^0_i(m) \) is called the imported signature and \( \mathcal{S}^0_e(m) \) is called the exported signature of \( m \). For \( m \in \text{AtSPEC} \) the element \( \mathcal{M}^0(m) \) denotes a partial function \( \text{Alg}_{\mathcal{S}^0_i(m)} \rightarrow \text{Alg}_{\mathcal{S}^0_e(m)} \) where \( \text{Alg}_{\Sigma} \) is the class of all \( \Sigma \)-algebras for a given signature \( \Sigma \). The inherited signature \( \mathcal{S}^0_i(m) \cap \mathcal{S}^0_e(m) \) for \( m \in \text{AtSPEC} \) is abbreviated as \( \mathcal{I}(m) \) and the set \( \text{SORT}(\mathcal{I}(m)) \) — the inherited sorts — as \( I(m) \). Moreover we assume that each \( \mathcal{M}^0(m) \) fulfills the following persistency condition:

\[
\text{for each } \mathcal{S}^0_i(m)\text{-algebra } A \text{ such that } \mathcal{M}^0(m)(A) \text{ is defined we have } \\
\mathcal{M}^0(m)(A)_s = A_s \text{ for } s \in I(m) \text{ and } \\
f \mathcal{M}^0(m)(A) = f_A \text{ for } s \in \mathcal{I}(m).
\]

The language \( \text{SPEC} \), whose elements will be denoted as specifications or modules henceforth, is defined as a subclass of the set inductively generated over \( \text{AtSPEC} \) using the specification building operations

\footnote{Here we differ from the more general logical framework sketched in [LL87] in restricting our considerations to first order logic. We have chosen this restriction because in the context of implementations we only prove results for this kind of formulas.}